Національний технічний університет України
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ
(повна назва інституту/факультету)

кафедра БІОМЕДИЧНОЇ КІБЕРНЕТИКИ
(повна назва кафедри)

«На правах рукопису»
УДК 004.021

«До захисту допущено»
Завідувач кафедри БМК

_____ Є.А. Настенко
(підпис) (ініціали, прізвище)

“___”_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 122 «Комп’ютерні науки та інформаційні технології»
(код і назва)

на тему: «Програмна система для порівняльного дослідження збіжності алгоритмів лінійної класифікації»

Виконав (-ла): студент (-ка) **VI** курсу, групи БС-61м
(шифр групи)

МАТУШЕВИЧ НАТАЛІЯ АНАТОЛІЇВНА _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник *проф. каф. БМК, проф., д.т.н. Файнзільберг Л.С.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з розділів МД _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент *доц. каф. БМІ, доц., к.т.н., Зубчук В.І* _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2018 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут (факультет) БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ
(повна назва)

Кафедра БІОМЕДИЧНОЇ КІБЕРНЕТИКИ
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-науковою програмою
спеціальність 122 «Комп'ютерні науки та інформаційні технології»
(спеціалізація) (Інформаційні технології в біології та медицині)
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри БМК
_____ Є.А. Настенко_
(підпис) (ініціали, прізвище)

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

МАТУШЕВИЧ НАТАЛІЇ АНАТОЛІЇВНІ
(прізвище, ім'я, по батькові)

1. Тема дисертації «Програмна система для порівняльного дослідження збіжності алгоритмів лінійної класифікації»

науковий керівник дисертації
Файнзільберг Леонід Соломонович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «_29_»_березня_ 2018 р. №_1041-с

2. Термін подання студентом дисертації ***11-12 травня 2018 року***
3. Об'єкт дослідження Алгоритми для лінійної класифікації об'єктів

4. Предмет дослідження Алгоритми Розенблата та Козинця для лінійної класифікації об'єктів, їх властивості

5. Перелік завдань, які потрібно розробити
Проаналізувати алгоритми лінійної класифікації об'єктів, програмно реалізувати обрані алгоритми, провести дослідження збіжності обраних алгоритмів, проаналізувати результати експериментів

6. Орієнтовний перелік графічного (ілюстративного) матеріалу

Графіки побудованого лінійного розділювача для кожного з алгоритмів, гістограма порівняння збіжності алгоритмів.

7. Орієнтовний перелік публікацій 1.Comparative evaluation of convergence's speed of learning algorithms for linear classifiers by statistical experiments method introduction// Кибернетика и вычислительная техника. – 2018. – № 2. 2. Огляд алгоритму Козинця для побудови оптимальної розділяючої гіперплощини // Актуальные научные исследования в современном мире // Сб. научных трудов 3. Порівняльний аналіз алгоритмів побудови розділяючої гіперплощини // Теорія і практика накових знань (частина IV)

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Магістерської дисертації			

9. Дата видачі завдання ***19 березня 2018 р.***

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримати завдання на МД	19 березня 2018р..	
2	Огляд літератури	30 березня 2018р.	
3	Побудова структури програмної системи	10 березня 2018р.	
4	Проведення експериментів	10 квітня 2018р.	
5	Написання МД	1травня 2018р.	
6	Предзахист МД та допуск до захисту дисертації	3 травня 2018р..	
7	Подання МД рецензенту. Отримання рецензії.	4-7 травня 2018р.	
8	Подання в електронному вигляді МД та анотації до неї на сайт кафедри.	11-12 травня 2018р.	
9	Подання пакету документів по МД до захисту в ЕК	11-12 травня 2018р.	
10	Захист МД в ЕК	18-19 травня 2018р..	

Студент _____ Матушевич Н.А.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Файнзільберг Л.С.
(підпис) (ініціали, прізвище)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ШНМ – штучні нейронні мережі.

RNN – a recurrent neural network (рекурентна нейронна мережа).

ПЗ – програмне забезпечення

ПП – програмний продукт

DFD – Data Flow Diagram

ВСТУП

Актуальність теми. Завдання навчання розпізнавання об'єктів різної фізичної природи (Mashine Leatning) - одна з головних задач штучного інтелекту [1-5]. Досить часто таке завдання розглядаються як завдання визначення параметрів дискримінантної функції (функцій) в багатовимірному просторі ознак [2].

Особливої уваги заслуговують лінійні дискримінантні функції, до яких, згідно [3], в просторі ймовірностей зводиться будь-яка байєсівська стратегія розпізнавання. При цьому слід мати на увазі, що лінійна дискримінантна функція передбачає, що збільшення значень однієї ознаки може бути компенсовано зменшенням значення іншої ознаки, що не завжди вірно [4]. Незважаючи на це лінійні класифікатори знаходять широке застосування при вирішенні багатьох практичних завдань [5 - 10].

У тих випадках, коли Природа йде назустріч конструктору прикладної системи і в вихідному або перетвореному просторі ознак об’єкти, які розпізнаються, можуть бути повністю розділені лінійною дискримінатною функцією завдання зводиться до навчання лінійного класифікатора за кінцевим числом спостережень [3]. Існують різні алгоритми навчання лінійних класифікаторів, два з яких - алгоритм навчання персептрона, запропонований Френком Розенблатом [5] і алгоритм Б.Н.Козинця [11].

У відомій теоремі Новикова доведено, що за умови лінійної роздільності навчальної вибірки алгоритм персептрона сходиться за кінцеве число ітерацій [3]. Аналогічна теорема про кінцеву збіжності доведена і для алгоритму Козинця [12].

У той же час формальні умови, які дають оцінку числа ітерацій вказаних алгоритмів, досить грубі [3, с. 190] і тому не дозволяють однозначно відповісти на важливе питання: який з алгоритмів і коли забезпечує більшу швидкість збіжності в процесі навчання по кінцевої

вибірці спостережень. Невідомі також і ряд інших властивостей названих алгоритмів, які важливі при вирішенні конкретних практичних завдань.

Мета роботи – розробити програмну систему для дослідження властивостей алгоритмів навчання Розенблата і Козинця на основі проведення статистичного експерименту методом Монте-Карло.

У відповідності до мети сформовано наступні **задачі**:

- провести аналіз алгоритмів лінійної класифікації об’єктів;
- створити програмну систему для зручного дослідження збіжності алгоритмів;
- програмно реалізувати алгоритм Розенблата;
- програмно реалізувати алгоритм Козинця;
- провести дослідження збіжності обраних алгоритмів;
- проаналізувати результати експериментів.

Об’єкт дослідження – алгоритми для лінійної класифікації об’єктів.

Предмет дослідження – алгоритми Розенблата та Козинця для лінійної класифікації об’єктів, їх властивості.

Для досягнення мети дослідження та реалізації програмної системи використано **методи** Microsoft Visual Studio 2017 з використанням мови програмування C#.

Реалізація результатів роботи.

Робота виконана на замовлення Міжнародного науково-навчального центру інформаційних технологій і систем Національної Академії наук України та Міністерства освіти і науки України. Одержані результати дослідження впроваджені в Міжнародному науково-навчальному центрі інформаційних технологій і систем Національної Академії наук України та Міністерства освіти і науки України (акт впровадження від 25.04.18).

Апробація результатів роботи.

Основні положення та результати магістерської дисертації були викладені в наступних роботах:

1. Comparative evaluation of convergence's speed of learning algorithms for linear classifiers by statistical experiments method introduction// Кибернетика и вычислительная техника. – 2018. – № 2.

2. Огляд алгоритму Козинця для побудови оптимальної розділяючої гіперплощини // Актуальные научные исследования в современном мире // Сб. научных трудов - Переяслав- мельницкий, 2017. - Вып. 12(32), ч. 7. С.122-125

3. Порівняльний аналіз алгоритмів побудови розділяючої гіперплощини // Теорія і практика накових знань (частина IV): матеріали II Міжнародної науково-практичної конференції м. Київ, 28-29 грудня 2017 року. – Київ.: МЦНД, 2017. С. 31-31.

Структура дисертації

Дисертація побудована за класичним типом та викладена на 143 сторінках машинописного тексту. Складається з вступу, 5 розділів, висновків, списку використаних літературних джерел, який містить 48 найменувань, 21 – на кирилиці, 27 – на латиниці та одного додатку з лістингом коду. У роботі представлено 56 рисунків і 7 таблиць.

РОЗДІЛ 1
ЗАГАЛЬНІ ВІДОМОСТІ З ПРЕДМЕТНОЇ ОБЛАСТІ

Сучасний світ потопає в морі інформації. Людина мануально не в змозі класифікувати гігабайти текстів та зображень. Для того щоб продовжувати працювати з великим об'ємом даних використовується комп'ютер і математичні алгоритми для вирішення повсякденних завдань.

На допомогу приходять нейронні мережі, а точніше наближена математична інтерпретація реальних нейронних мереж, котрі працюють в людському мозку. Оскільки математичне уявлення нейронної мережі це рівняння з N кількістю невідомих і наше завдання підібрати такі коефіцієнти в цьому рівнянні для кожної невідомої щоб в середньому відповідь була наближена до правильної. Такі мережі не треба програмувати в звичному розумінні цього слова, а треба навчати. Що в свою чергу дає можливість використовувати одну й ту ж саму структуру мережі для вирішення різних завдань [6].

Головним завданням яке вирішує нейронна мережа є завдання класифікації, але в залежності від того як будуть інтерпретуватися результати виконання функції з’являється можливість вирішувати інші завдання в реальному світі, які з першого погляду не дуже схожі на завдання класифікації.

Припустимо, що є мережа яку навчили розпізнавати образи рукописних цифр і вона успішно може застосовуватися на пошті для швидкого розпізнання індексу на листах. А тепер уявімо, що постало завдання розпізнавати голосові команди. Як це можливо зробити? Здавалося б, що звук це хвиля, а не статична картинка і на перший погляд нам потрібна мережа з іншою структурою. Зовсім не обов'язково брати іншу реалізацію мережі, звук можливо представити як картинку спектра звукової хвилі і навчити розпізнавати картинки спектра. Тепер ми можемо після навчання цієї мережі подавати їй на вхід картинку спектра і на виході

отримувати відповідь до якого класу належить ця картинка. Таким чином, ми отримали мережу яка може розпізнавати голосові команди [6].

Як же навчаються мережі? Основним і загальноприйнятим алгоритмом навчання є метод градієнтного спуску. Метод, який дозволяє підібрати такі коефіцієнти рівняння де середньоквадратична помилка буде мінімальною. Тобто буде вирішуватися класична задачу оптимізації та мінімізацію функції помилки. Такий метод дуже ефективний, але у нього є обмеження. Наша функція (мережа) протягом усієї області значень може мати кілька мінімумів і в залежності від початкових значень і швидкості навчання є можливість потрапити в локальний мінімум в якому задача вирішується на допустимому рівні, але є такий мінімум який дозволить зменшити помилку мережі. Для того, щоб вирішення було оптимальним необхідно підібрати правильні параметри швидкості навчання і початкові значення [7].

Є інший спосіб вирішення подібних завдань оптимізації, з застосуванням генетичних алгоритмів. Оскільки генетичний алгоритм це евристичний метод пошуку наших коефіцієнтів для мережі, то він допоможе нам зменшити ймовірність попадання в локальні мінімуми функції.

1.1. Історія появи нейронних мереж

Ідея створення «мислячої» машини - принаймні така ж стара, як сучасна обчислювальна техніка, навіть ще старша. Алан Тьюринг у своєму основному документі «Обчислювальна техніка та інтелект» виклав кілька критеріїв, щоб визначити, чи можна назвати машину розумною, яка з того часу стала відома як «тест Тьюрінга». Для деяких чудових досліджень на варіантах тесту Тьюринга можна виділити книгу Браяна Крістіана, в якій докладно описані його пригоди з премією Лобнера під назвою «The Most Human Human»[6].

Рання робота в галузі машинного навчання була в основному інформована поточними робочими теоріями мозку. Першими хлопцями на сцені були Вальтер Піттс і Уоррен МакКуллок. Вони розробили техніку, відома під назвою «thresholded logic unit», яка була розроблена, щоб імітувати спосіб роботи нейрона[7].

Продовжуючи, необхідно сказати, що «персептрон» Френка Розенблата, це перший справжній попередник сучасних нейронних мереж. В той час ця річ була досить вражаючою. Воно прийшло з процедурою навчання, яка навмисно зближалася б з правильним рішенням і могла б розпізнати букви та цифри. Розенблат був настільки впевнений, що персептрон приведе до справжнього штучного інтелекту, що в 1959 році він зауважив[7]:

«[Персептрон] - ембріон електронного комп'ютера, який, як очікується, зможе ходити, говорити, бачити, писати, відтворювати себе і усвідомлювати його існування.»

Персептрон Розенблата почав привертати досить багато уваги, і особливо увагу однієї особи. Марвін Мінський, чкого часто вважають одним з батьків штучного інтелекту, почав відчувати, що щось невистачає в персептроні Розенблата. Мінський казав[6]:

«Однак я почав турбуватися про те, що тій машині не вдалося зробити. Наприклад, він міг би відрізнити «E» від «F», а «5» від «6» – щось на кшталт цього. Але коли поруч з цими фігурами, які не співвідносилися з ними, були подразники, шуми, то визнання не відбувалося.»

Разом з кандидатом наук, Сеймором Папертом, Мінкі написав книгу під назвою «Персептрони», яка ефективно вбила персептрон, закінчивши ембріональну ідею нейронної мережі. Вони показали, що персептрон не вмів навчатися простій XOR функції. Що найгірше, вони довели, що теоретично це неможливо, незалежно від того, як довго ви її навчаєте. Тепер це нам не дивно, тому що модель, що має на увазі персептрон, є

лінійною, а функція XOR нелінійна, але в той час це було достатньо для того, щоб убити всі дослідження на нейронних мережах[6].

Дуже важко злитися на Мінського, і, він звучить як той, хто наближається до платонічного ідеалу вченого та академічної людини. Ось цитата з статті:

«Коли він був студентом, він сказав, що для нього з'явилися, як мінімум, лише три цікавих проблеми в світі або в світі науки. «Генетика, здавалося, була досить цікавою, тому що ще ніхто не знав, як це працює», - сказав він. «Але я не був упевнений, що це глибоко. Проблеми фізики здавалися глибокими і вирішеними. Можливо було б добре вивчати фізику. Але проблема інтелекту здавалася безнадійно глибокою.»

Член правління Джордж Баул, людина на ім'я Джефф Хінтон, закінчив свою ступінь доктора філософії з вивчення нейронних мереж у 1978 році, а до 1986 року переїхав до інституту пізнавальної науки в Університеті Сан-Дієго. Окрім Девіда Румелхарта та Рональда Вільямса, Хінтон опублікував статтю під назвою «Навчальні уявлення про побічні помилки». У цій роботі вони показали, що нейронні сітки з багатьма прихованими шарами можуть бути ефективно навчені відносно простою процедурою. Це дозволить нейронним мережам подолати слабкість персептона, оскільки додаткові шари наділяють мережу здатністю вивчати нелінійні функції. Приблизно в той же час було показано, що такі мережі мали здатність вивчати будь-яку функцію, результат, відомий як теорема про загальну апроксимацію[6].

Вивченню людського мозку - тисячі років. З появою сучасної електроніки було природним спробувати використовувати цей процес мислення [6]. Перший крок до штучних нейронних мереж прийшов в 1943 році, коли Уоррен Маккаллох, нейрофізіолог і молодий математик Уолтер Піттс, написав статтю про те, як нейрони можуть працювати. Вони змоделювали просту нейронну мережу з електричними схемами[6].

Зміцнення цієї концепції нейронів і їх роботи - це книга, написана Дональдом Хеббом. Організація поведінки була написана в 1949 році [6]. Вона вказала, що нейронні шляхи зміцнюються кожен раз, коли вони використовуються[6].

У міру того, як комп'ютери просувалися на початку 50-х років, стало можливим приступити до моделювання зачатки цих теорій, що стосуються людської думки. Натаніель Рочестер з дослідних лабораторій IBM вперше зробив спробу імітувати нейронну мережа. Ця перша спроба не вдалася, але подальші спроби були успішними [7]. Саме в цей час традиційні обчислення почали розквітати.

У 1956 році Дартмутський дослідний проект з штучного інтелекту дав імпульс як штучному інтелекту, так і нейронним мереж.

У роки, наступні за Дартмутського проектом, Джон фон Нейман запропонував наслідувати простим функціям нейронів, використовуючи телеграфні реле або вакуумні трубки. Крім того, Франк Розенблат, нейробіолог Корнелла, почав роботу над персептроном. Він був заінтригований роботою ока мухи. Велика частина обробки, яка каже мусі бігти, робиться в її оці. Персептрон, що виник в внаслідок цього дослідження, був побудований на апаратному забезпеченні та є найстарішою нейронною мережею, яка все ще використовується сьогодні. Було виявлено, що одношаровий персептрон корисний при класифікації безперервного набору вхідних даних в один з двох класів. Персептрон обчислює зважену суму входів, віднімає поріг і як результат видає одне з двох можливих значень [6]. На жаль, персептрон має обмеження, що було доведено в книгах Первітронов Марвіна Мінскі і Сеймура паперті 1969 року. Він не здатній вирішувати певний клас завдань пов'язаних з інваріантністю уявлень. Інтерес до цієї галузі науки різко згасає.

У 1982 році кілька подій викликали новий інтерес. Джон Хопфилд з Caltech представив доповідь американської національної академії наук. Підхід Хопфілда складався не в просто моделюванні мізків, а в створенні

корисних пристроїв. З ясністю і математичним аналізом він показав, як можуть працювати такі мережі і що вони можуть робити [7]. Проте, найбільшим активом Хопфилда була його харизма.

І 1986 році Американська група вчених і Красноярська група вчених незалежно і одночасно перевідкривають алгоритм зворотного поширення помилки, що призводить до різкого зростання інтересу до неронних мереж, які навчаються [6].

1.2. Визначення нейронних мереж

Нейронна мережа - це система, яка здатна приймати рішення на основі великої кількості умов, які надходять на вхід та видавати відповідь на виході [6].

Нейрон – це нервова клітина, що входить до складу біологічної системи. Він складається з тіла і великої кількості відростків, які слугують його зв1язком із зовнішнім світом (рис. 1.1).

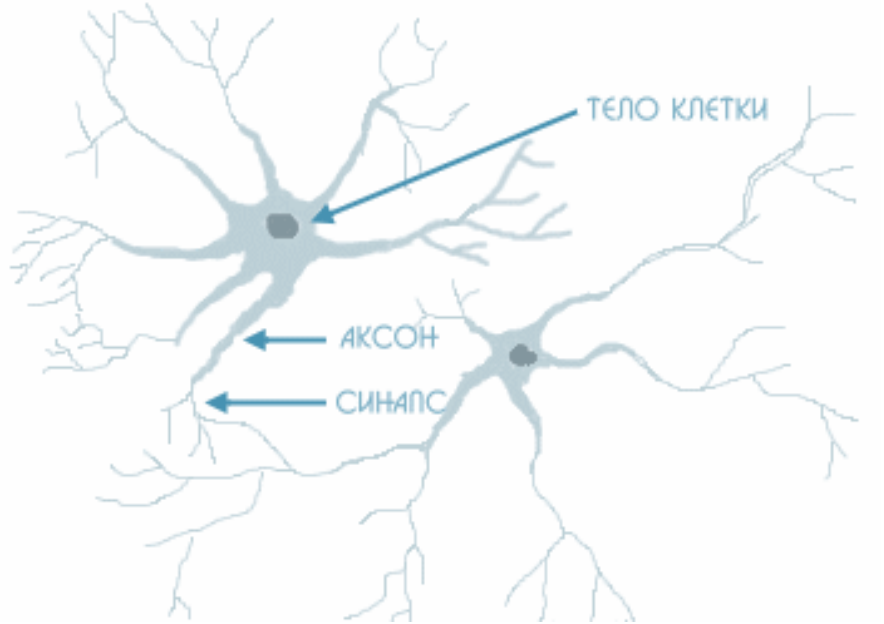


Рисунок 1.1. Біологічний нейрон.

Дендритами називаються ті відростки, по яким нейрон отримує збудження [7]. Після цього нейрон передає збудження по унікальним

відросткам – аксонам. У кожного нейрона лише один аксон. Місце з'єднання аксона нейрона - джерела збудження з дендритом називається синапсом [7]. Основна функція нейрона полягає в передачі збудження з дендритів на аксон. Але сигнали, що надходять з різних дендритів, можуть надавати різний вплив на сигнал в аксоні. Якщо все збудження, що надыйшло перевищить деяке порогове значення, яке в загальному випадку змінюється в деяких межах, то нейрон видасть сигнал [7]. В іншому випадку на аксон сигнал виданий не буде: нейрон не відповість на збудження. В цій загальній схемі дуже багато ускладнень і винятків, проте, переважна більшість штучних нейронних мереж здатні моделювати лише ці прості властивості [7].

Патерсон в 1996 році намагався відтворити здатність до навчання та виправлення помилок в біологічних нервових системах за допомогою моделювання низькорівневої структури мозку. Саме з цих досліджень і виникли нейронні мережі. Основною областю досліджень з штучного інтелекту в 60-ті - 80-ті роки були експертні системи [8]. Такі системи ґрунтувалися на моделюванні процесу мислення та навчання (зокрема, на тому, як побудований на маніпуляціях з символами процес нашого мислення) [8]. Скоро стало зрозуміло, що подібні системи, хоча і можуть принести користь в деяких областях, але не в змозі проникнути до багатьох ключових аспектів людського інтелекту [8]. Згідно з однією з точок зору [8], причина такого явища полягає в тому, що вони не в змозі відтворити структуру мозку. Для створення штучного інтелекту необхідно побудувати систему схожу архітектурою на людський мозок.

Мозок складається з надзвичайно великого числа (приблизно 10,000,000,000) нейронів, котрі з'єднані за допомогою численних зв'язків (кілька тисяч зв'язків на один нейрон, в середньому, проте це число може дуже змінюватися). Під час активації нейрон посилає електрохімічний сигнал по своєму аксону [6]. За допомогою синапсів цей сигнал досягає інших нейронів, які в свою чергу активуються. Нейрон активується в тому

випадку, коли сумарна кількість сигналів, які надійшли до його ядра з дендритів, перевищить певний рівень (поріг активації) [6].

Інтенсивність цього сигналу, котрий одержав нейрон (і можливість його активації), сильно залежна від активності синапсів нейрона. Кожен синапс має протяжність, і хімічні речовини передають сигнал вздовж нього. Дональд Хебб, котрий є одним з найавторитетніших дослідників в області нейросистем, висловив постулат про те, що навчання полягає в першу чергу в змінах «сили» синаптичних зв'язків. Наприклад, в класичному досліді Павлова, собака швидко навчилася пов'язувати дзвінок дзвоника з їжею, оскільки кожен раз безпосередньо перед годуванням собаки дзвонив дзвоник. Синаптичні зв'язки між ділянками кори головного мозку, які відповідають за слух, і слинні залози посилилися, і при збудженні кори звуком дзвіночка у собаки починалося слиновиділення [6].

Таким чином, мозок здатний вирішувати надзвичайно складні завдання, оскільки його побудовано з дуже великого числа абсолютно простих елементів (кожен з яких приймає зважену суму вхідних сигналів і в разі, якщо сумарний вхід перевищує певний рівень, передає далі двійковий сигнал). Зрозуміло, що тут не говорилося про багато складних аспектів будови мозку, однак цікаво те, що штучні нейронні мережі, використовуючи модель, яка не набагато складніше, ніж описана вище, мають можливість досягати чудових результатів [6].

1.2.1. Штучний нейрон

Штучні нейронні мережі (ШНМ) – це математичні моделі, а також їх апаратні або програмні реалізації, які побудовано за принципом функціонування та організації біологічних нейронних мереж, тобто мереж нервових клітин живих організмів. Ці поняття виникли під час вивчення процесів, які мали місце в мозку живого організму, та під час спроб моделювання таких процесів. Нейронна мережа, які були побудовані Маккалоком і Парсом стали першою такою спробою. Згодом, після того як

були розроблені перші алгоритми навчання, одержані моделі стали використовуватися для практичних цілей, а саме: в задачах пов’язаних з розпізнаванням образів, прогнозуванням та ін [7].

ШНМ – це системи простих штучних нейронів (процесорів), котрі взаємодіють та з'єднані між собою. В порівнянні з процесорами, котрі використовуються всередині персональних комп'ютерів, ці процесори зазвичай є досить простими. Кожен процесор такої мережі має справу тільки з сигналами, які він періодично отримує (вхідними), і сигналами, які він періодично посилає іншим процесорам (вихідними). І, тим не менше, такі частково прості процесори поєднані разом здатні до виконання складних завдань, оскільки з'єднані в досить великі мережі з керованими взаємодіями [8].

Окремим випадком методів розпізнавання образів слугують нейронні мережі, з точки зору машинного навчання, а саме: дискримінантного аналізу, методів кластеризації і т. п. Навчання нейронних мереж – це багато параметричне завдання нелінійної оптимізації, з математичної точки зору. З кібернетичної точки зору, нейронні мережі використовуються в задачах адаптивного управління і як робото технічні алгоритми. Якщо розглядати розвиток обчислювальної техніки та програмування, то нейронні мережі – спосіб знаходження розв’язку проблеми ефективного паралелізму [7]. А з точки зору штучного інтелекту, ШНМ являє собою основу філософської течії конективізму і слугує основним напрямком в структурному підході з вивчення можливості побудови (моделювання) природного інтелекту використовуючи комп'ютерні алгоритми [7].

Нейронні мережі навчають, а не програмують в звичному сенсі цього слова. Те, що вони здатні навчатися – це одна з головних переваг нейронних мереж в порівнянні з традиційними алгоритмами. З технічної точки зору навчання полягає в знаходженні коефіцієнту кожного зі зв'язків між нейронами. Під час навчання нейронна мережа може ідентифікувати

складні залежності між вхідними вибірками і вихідними, а також виконувати узагальнення. Це означає, що, якщо навчання пройшло успішно, мережа здатна дати вірну відповідь на підставі даних, які були відсутні в навчальній вибірці, а також неповних і / або «зашумлених», чи частково змінених даних [8].

Штучний нейрон являє собою імітацію властивостей біологічного нейрону в першому наближенні. На вхід штучного нейрона подають деяку множину сигналів, кожен з яких слугує виходом іншого нейрона. Кожен із входів множиться на відповідну йому вагу, аналогічну синаптичній силі, і всі множники підсумовуються, визначаючи рівень активації нейрона. На рис. 1.2 можна побачити безліч вхідних сигналів, що надходить на штучний нейрон. Ці вхідні сигнали, в сукупності, що позначені вектором *X*, відповідають сигналам, що приходять до синапсів біологічних нейронів [7]. Кожен сигнал множиться на відповідну вагу і надходить на блок сумування. Кожна вага ставиться у відповідність до «сили» одного з біологічних синаптичних зв'язків. Сумуючий блок, що відповідає тілу біологічного елементу, сумує алгебраїчно зважені входи, створюючи вихід [7].

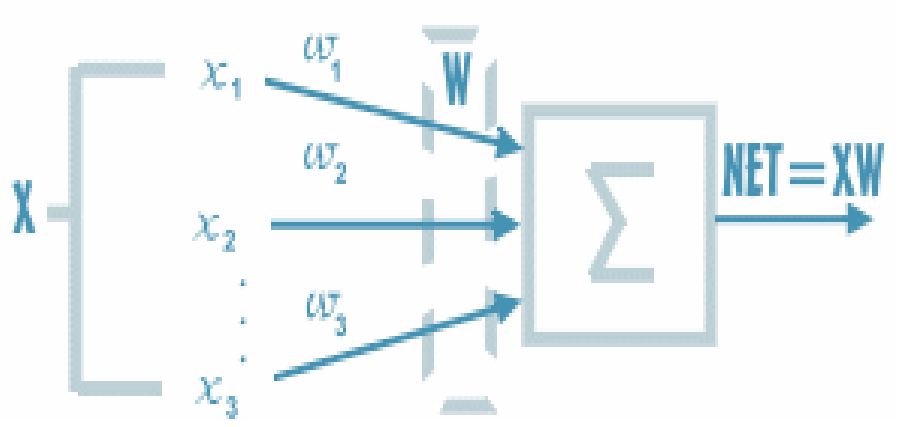


Рисунок 1.2. Штучний нейрон.

Хоч лише один з нейронів і здатен виконати найпростішу процедуру розпізнавання, вся потужність нейронних обчислень полягає у з'єднанні нейронів в мережі [7].

Більші і складніші нейронні мережі можуть мати, як правило, і більші обчислювальні можливості. Хоча створено мережі всіх можливих конфігурацій, які лише можливо собі уявити, пошарова організація нейронів такі структури деяких відділів мозку людини [8]. Багатошарова мережа може утворюватися за допомогою каскадів прошарків. Вихід одного слугує входом наступного шару [8].

Всі нейрони штучних нейронних мереж розділяються на підмножини, так звані шари. Взаємодія нейронів також відбувається пошарово [8].

Шари штучної нейронної мережі - це безліч нейронів, на які в кожен момент часу надходять паралельні сигнали від інших нейронів тієї ж мережі [8].

Завдання нейронної мережі визначає вибір архітектури. Для деяких типів задач вже виявлено оптимальні конфігурації. розробнику доводиться вирішувати задачу синтезу нової конфігурації тоді, коли завдання не може звестися ні до одного з відомих класів. Проблеми синтезу штучної нейронної мережі дуже залежать від завдань, для яких вони створюються [8]. У загальній більшості можливих випадків оптимальним варіантом штучної нейронної мережі є виявлений дослідним шляхом [8].

Штучна нейронна мережа може бути виконана програмно і апаратно. Реалізовані апаратно зазвичай являють паралельні обчислювачі, які складаються з безлічі простих процесорів [8].

1.2.2. Застосування нейронних мереж

Комп'ютери на основі нейронних мереж, або нейрокомп'ютери мають дуже багато властивостей, які є привабливими якщо розглядати їх практичне застосування [8]:

- надвисока швидкодія у зв’язку з використанням паралелізму під час обробки інформації;
- терпимість до помилок, тобто працездатність зберігається навіть під час пошкодження значного числа нейронів;

- здатність навчатися, тобто навчання приходить на зміну програмуванню нейронної мережі;
- здатність до розпізнавання образів в умовах сильних перешкод, шумів і спотворень [8].

Проте, перші дві з властивостей актуальні тільки для апаратної реалізації нейронних мереж. Такі нейронні мережі можуть забезпечувати знаходження результату для складних завдань за час, який необхідний для проходження всіх ланцюжків з електронних і / або оптичних елементів. Вихід майже не залежить від поломки окремого нейрона. Це обумовлює їх привабливість для використання в складі бортової обчислювальної системи [8].

Зараз нейронні мережі знайшли своє застосування під час вирішення багатьох завдань, які важко піддаються формалізації [7]:

- розпізнавання і синтезу мови;
- розпізнавання аерокосмічного зображення;
- прогнозування ринку цінних паперів і курсів валют;
- попередження махінацій за допомогою кредитних карток;
- оцінка вартості нерухомості;
- оцінка фінансового стану підприємств
- оцінка ризику неповернення кредитів;
- обробка сигналів радіолокації;
- контроль руху на швидкісних автомагістралях і залізницях;
- діагностика в медичній сфері;
- видобуток знань з великих обсягів даних в наукових дослідженнях, фінансах і бізнесі [7].

Нейронні мережі використовуються під час таких умов:

- якщо завдання можуть вирішувати люди;
- якщо під час вирішення завдання можна виділити велику множину вхідних факторів (сигналів, ознак, даних і т.п.) і велику множину вихідних факторів;

- якщо зміна вхідних сигналів призведе до зміни вихідних [8].

У той самий час застосування нейронних мереж при вирішенні деяких завдань може бути більш ефективним ніж використання розумових ресурсів людей. Це можна пояснити тим, що розум людини орієнтовано на вирішення завдань тривимірного простору. Багатовимірні завдання для нього є значно більшою трудомісткими [5]. Таке обмеження не властиве штучним нейронним мережам. Їм все одно вирішувати тривимірне або 10-мірне завдання [5].

При використанні нейронних мереж необхідно знайти вирішення таких завдань:

- постановка завдання, придатної для вирішення за допомогою нейронної мережі.
- вибір моделі ШНМ;
- підготовка вихідних даних для навчання ШНМ;
- навчання ШНМ;
- власне рішення задачі за допомогою навченої ШНМ;
- крім того, іноді потрібен ще один етап - інтерпретація рішення, отриманого нейронною мережею [5].

Найбільш трудомісткими процесами при використанні нейронних мереж є підготовка вихідних даних для навчання і навчання нейронної мережі [8].

1.3. Класифікація нейронних мереж

1.3.1. Нейронна мережа прямого поширення

Нейронна мережа прямого поширення є алгоритмом класифікації взятим з біології [7]. Вона складається з деякої (можливо великого) кількості простих нейроподібних блоків обробки, організованих в шарах. Кожна одиниця в шарі пов'язана з усіма одиницями в попередньому шарі. Ці з’єднання не всі рівні: кожне з'єднання може мати різну силу або вагу.

Ваги на цих з'єднаннях кодують знання мережі. Часто одиниці в нейронної мережі також називаються вузлами [7].

Дані надходять на входи і проходять через мережу, за шаром, до тих пір, поки вони не надійдуть на виходи. Під час нормальної роботи, тобто коли вона діє як класифікатор, між шарами немає зворотної зв'язку [8]. До таких мереж відносяться, наприклад: найпростіший персептрон (Розроблений Розенблатта) і багатошаровий перцептрон. На рис. 1.3 представлена логічна схема перцептрону з двома виходами [8].

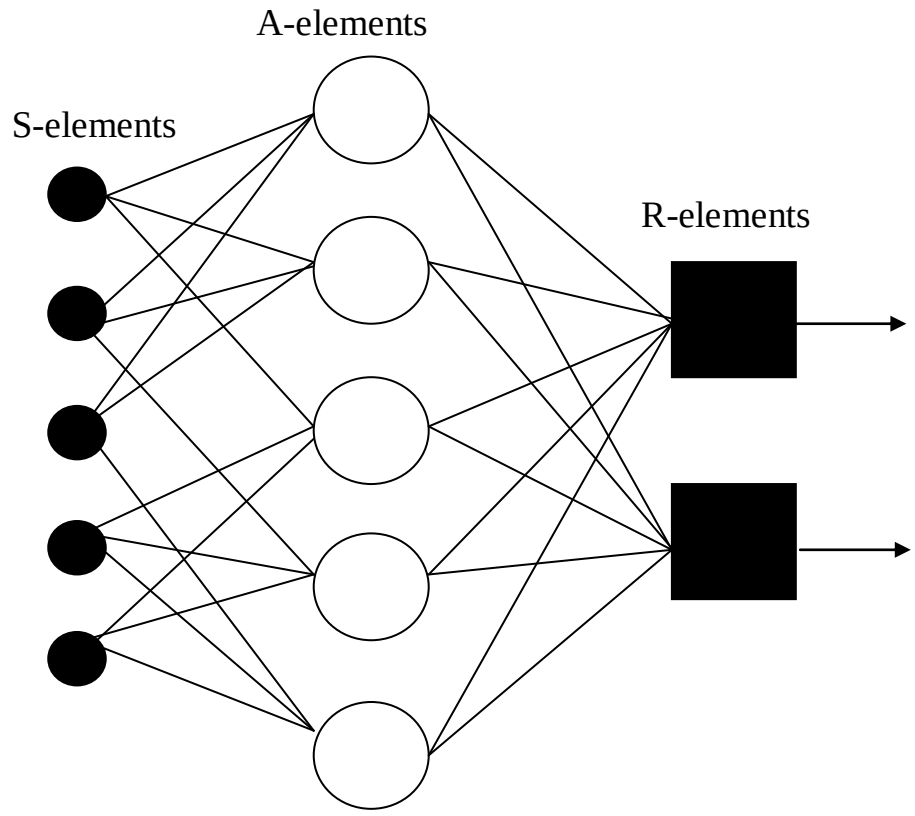


Рисунок 1.3. Одношаровий персептрон Розенблатта.

1.3.2. Рекурентна нейронна мережа

Рекурентна нейронна мережа (RNN) являє собою клас штучної нейронної мережі, де зв'язки між нейронами утворюють спрямований цикл. Це створює внутрішній стан мережі, який дозволяє їй демонструвати динамічну тимчасову поведінку [7]. На відміну від нейронних мереж з прямим зв'язком, RNN можуть використовувати свою внутрішню пам'ять

для обробки довільних послідовностей входів. Це дає можливість застосовувати їх до завдань, таких як не сегментоване розпізнавання рукописного введення або розпізнавання мови (рис. 1.4) [7].

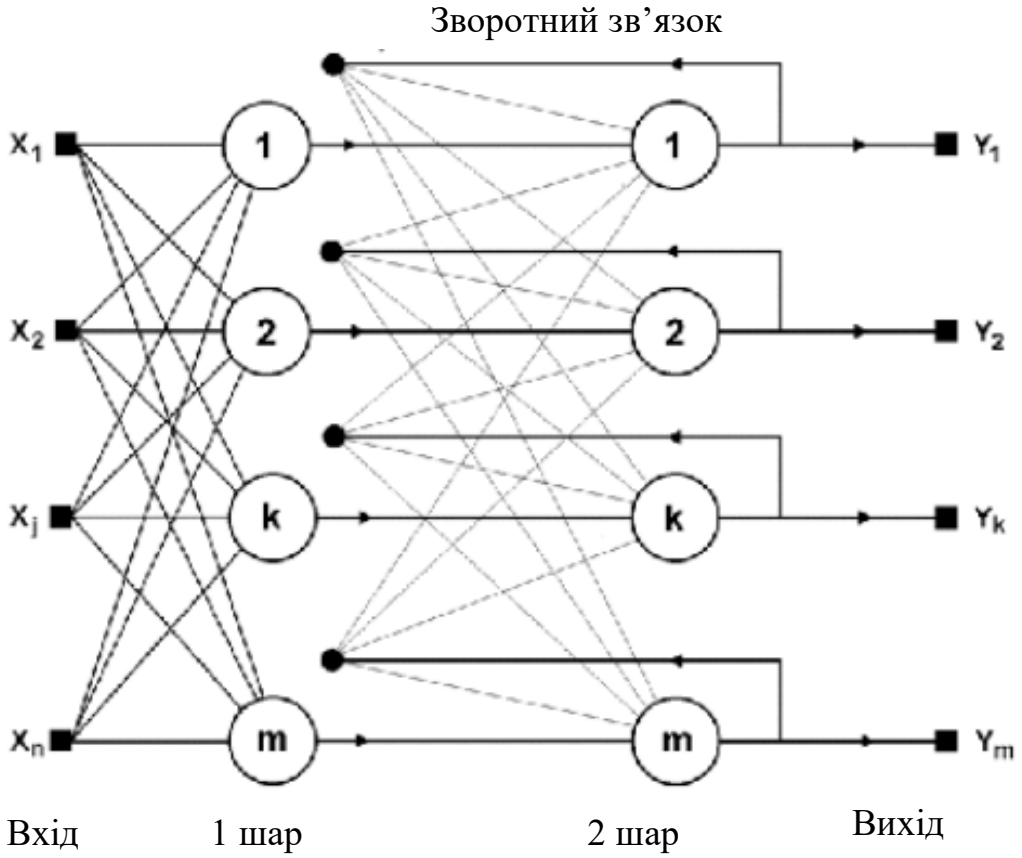


Рисунок 1.4. Схема рекурентної мережі.

1.3.3. Радіально-функціональна мережа

Радіально-функціональна мережа (рис. 3) - штучна нейронна мережа, яка використовує радіальну функцію як функцію активації [8]. Вихідний сигнал мережі являє собою лінійну комбінацію радіальних базисних функцій входів і параметрів нейронів. Радіальні базові функціональні мережі мають багато застосувань, включаючи апроксимацію функцій, прогнозування часових рядів, класифікацію і системний контроль [8].

У цій мережі є деякі особливості:

- вона має один прихований шар;
- тільки нейрони прихованого шару мають нелінійну активацію;

- синаптичні ваги між вхідним і прихованим шаром рівні 1 [8].

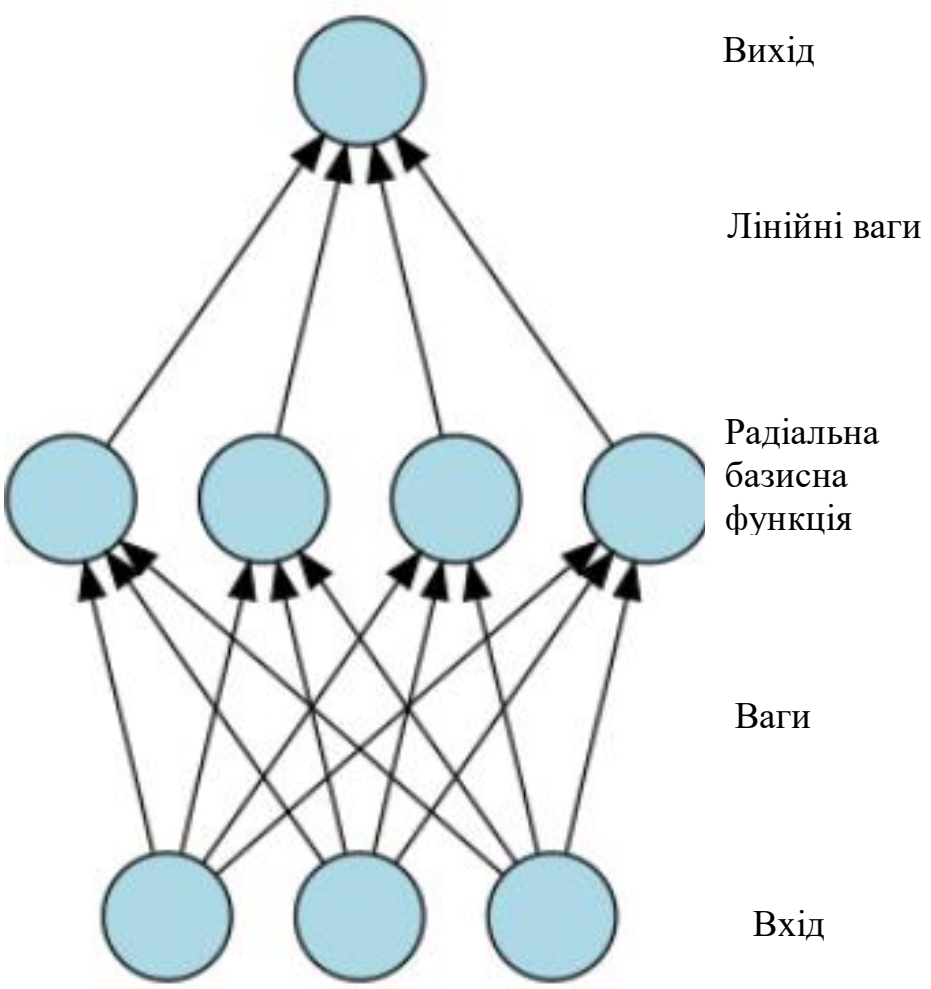


Рисунок 1.5. Архітектура мережі радіальних базисних функцій.

1.3.4. Самоорганізовані карти

Самоорганізована карта або самоорганізована карта функцій – це тип штучної нейронної мережі, яка навчається з використанням неконтрольованого навчання (метод без вчителя) для створення низькорозмірного (зазвичай двовимірного) дискретизованого уявлення. Самоорганізовані карти [8] відрізняються від інших штучних нейронних мереж тим, що вони застосовують конкурентну навчання на відміну від навчання корекції помилок (зворотне поширення помилки), вони використовують функцію околиці для збереження топологічних властивостей вхідного простору.

Самоорганізаційна карта складається з компонентів, які називаються вузлами або нейронами. Пов'язаним з кожним вузлом є вектор ваги тієї ж розмірності, що і вектори вхідних даних, і позиція в просторі карти. Звичайне розташування вузлів – це двовимірна регулярна відстань в гексагональній або прямокутній сітці. Самоорганізована карта описує відображення з простору з великими розмірами в простір з більш низьким розмірним відображенням [7]. Процедура розміщення вектора з простору даних на карту полягає в тому, щоб знайти вузол з найближчою (найменшою відстанню) ваговим вектором до вектору простору даних [7].

1.4. Методи навчання нейронних мереж

1.4.1. Метод навчання з «учителем»

Метод навчання з «учителем» - це задача машинного навчання виведення функції на підставі відомих даних навчання [7].

Дані навчання складаються з набору прикладів навчання. В данному методі навчання кожен приклад являє собою пару, що складається з вхідного об'єкта (зазвичай вектора) і бажаного вихідного значення (Також званого контрольним сигналом) [7]. Необхідно прорахувати значення функції на вхідних даних, потім порівняти з очікуваними, знайти помилку і потім скорегувати ваги мережі на цю помилку [7].

Алгоритм навчання з «учителем»:

1. Взяти дані для навчання мережі і розділити їх на дві частини. Перша це навчальна вибірка, друга це тестова вибірка. (можна розділити дані 70/30);

2. Порахувати значення функції для навчальної вибірки і знайти значення функції помилки;

3. Скорегувати ваги синапсів всередині мережі;

4. Повторювати пункти 2 і 3 до тих пір поки значення функції помилки не буде мінімальна. Повторювати ці дії можна як для кожного

примірника навчальної вибірки так і для всієї вибірки в цілому. У першому випадку навчання буде повільніше, але з більшою точністю. У другому, ми жертвуємо точністю, але навчання буде відбуватися швидше;

5. Перевірити всі значення за тестовою вибіркою, щоб зрозуміти наскільки добре система змогла узагальнити дані (навчитися) [7].

До алгоритмів навчання з учителем відносять:

1. Алгоритми локальної оптимізації з обчисленням часткових похідних першого порядку:

- градієнтний алгоритм (метод найшвидшого спуску);
- методи з одновимірної і двовимірної оптимізацією цільової функції в напрямку антиградієнта;
- метод спряжених градієнтів;
- методи, що враховують напрямок антиградієнта на декількох кроках алгоритму [7].

2. Алгоритми локальної оптимізації з обчисленням часткових похідних першого і другого порядку:

- метод Ньютона;
- методи оптимізації з розрідженими матрицями Гессе;
- квазіньютоновскі методи;
- метод Гаусса-Ньютона;
- метод Левенберга-Марквардта і ін [7].

3. Стохастичні алгоритми оптимізації:

- пошук у випадковому напрямку;
- імітація відпалу;
- метод Монте-Карло (чисельний метод статистичних випробувань) [7].

4. Алгоритми глобальної оптимізації (задачі глобальної оптимізації вирішуються за допомогою перебору значень змінних, від яких залежить цільова функція) [7].

Серед зазначених вище алгоритмах найбільш популярний градієнтний алгоритм і зокрема метод зворотного поширення помилки [7].

Метод зворотного поширення помилки.

Метод зворотного поширення помилки є поширеним методом навчання штучних нейронних мереж і використовується в поєднанні з методом оптимізації, таким як градієнтний спуск. Алгоритм повторює двофазний цикл, поширення та оновлення ваги. Коли вхідний вектор представляється в мережу, він поширюється по мережі через шар, поки не досягне вихідного рівня. Потім вихід мережі порівнюється з бажаним виходом, використовуючи функцію втрат, і обчислюється значення помилки для кожного з нейронів в вихідному шарі. Значення помилок потім поширюються назад, починаючи з виходу, до тих пір, поки кожен нейрон не матиме відповідне значення помилки, яке приблизно відповідає його внеску в вихідний висновок [8].

Даний метод можна використовувати для кожного елемента навчальної вибірки, так всієї вибірки в цілому (пакетний метод). У першому випадку збіжність функції буде повільніше, але точність вище. У другому все навпаки [8].

Є кілька моментів які треба врахувати при навчанні даними методом:

- число прикладів має бути вище числа ваг в мережі;
- можлива перетренировка мережі - чим менше помилка після навчання на прикладах тим гірше генералізація;
- у мережах в зворотними зв'язками збіжність значно гірше [8].

1.4.2. Метод навчання «без вчителя»

Цей метод також називає методом неконтрольованого навчання, яке зможе знайти структуру або відносини між різними входами [6]. Головна відмінність від методу навчання "з вчителем", це те що ми маємо тільки вхідні дані. Найбільш важливим неконтрольованим навчанням є кластеризація, яка створить різні кластери введення і зможе вносити будь-які нові дані до відповідного кластеру. Крім кластеризації, інші методи

неконтрольованого навчання це: виявлення аномалій, навчання і навчання в теорії Хебба. Приховані змінні моделі, такі як алгоритм максимізації очікувань, метод моментів (середня, ковариация) і методи поділу сліпих сигналів (Аналіз основних компонентів, незалежний аналіз компонентів, негативна матрична факторизація, сингулярне розкладання) [7].

Для вирішення різних завдань є готові підходи до навчання:

1. Завдання кластеризації:

- нейронна мережа Кохонена;
- графові алгоритми кластеризації;
- метод найближчих сусідів;
- інші [7].

2. Завдання скорочення розмірності:

- багатомірне шкалювання;
- метод головних компонент;
- інші [7].

3. Завдання візуалізації даних

- карта Схожості;
- самоорганізована карта Кохонена;
- дендрограмма [7].

Для роботи із зображеннями можна застосовувати метод найближчих значень для зменшення вихідного зображення. За допомогою даного методу можна отримати так звані фільтри (ядра згортки) для сверточних нейронних мереж [7].

Метод найближчих сусідів (k-means)

Даний метод являє собою метод векторного квантування [8]. Метод найближчих сусідів спрямований на поділ n спостережень на k кластерів, в яких кожне спостереження належить кластеру з найближчим середнім значенням, службовцям прототипом кластера [8]. Це призводить до розбиття простору даних на осередки.

Термін «k-means» був вперше використаний Джеймсом Маккуїном в 1967 році, хоча ідея перегукується з Хьюго Штайнхаузу в 1957 році. Стандартний алгоритм був вперше запропонований Стюартом Ллойдом в 1957 році як метод імпульсно-кодової модуляції, хоча він не був опублікований за межами Bell Labs до 1982 року [8]. У 1965 році Е. В. Форг опублікував по суті той же метод, тому його іноді називають Ллойд-Фордж [8].

Переваги методу:

- простота реалізації;
- рішення є не гарантоване вірне, а найкраще з можливих [8].

Недоліки:

- неможливо сказати на якій підставі будуються відповіді, так як
- немає моделей або правил;
- обчислювальна трудомісткість;
- складність вибору заходи «близькості» [8].

1.5. Генетичні алгоритми

У 1950 році Алан Тьюринг запропонував «навчальну машину», яка буде паралельна принципам еволюції. комп'ютерне моделювання еволюції почалося ще в 1954 році з роботи Нільса Алла Баррічеллі, який використовував комп'ютер в Інституті перспективних досліджень в Прінстоні, штат Нью-Джерсі [8]. Його публікація 1954 року не отримала широкого розповсюдження. Починаючи з 1957 року австралійський генетик Алекс Фрейзер опублікував серію робіт по моделювання штучного відбору організмів. Комп'ютерне моделювання еволюції біологами стало більш поширеним на початку 1960-х років, і методи були описані в книгах Фрейзера і Бернелл (1970) і Кросбі (1973). моделювання Фрейзера включало в себе всі істотні елементи сучасних генетичних алгоритмів. Крім того, Ханс-Йоахім Бремерманн опублікував серію робіт в 1960-х

роках, в яких також поулярним було вирішення проблем оптимізації, що піддаються рекомбінації, мутації і селекції. Дослідження Бремермана також включало елементи сучасних генетичних алгоритмів [8].

Хоча Баррічеллі в роботі, про яку він повідомляв в 1963 році, змоделював еволюцію здатності грати в просту гру, штучна еволюція стала широко визнаним методом оптимізації в результаті роботи Інго Рехенберге і Ганса-Поля Швефеля в 1960-х роках І на початку 1970-х років група Решенберга змогла вирішити складні інженерні завдання за допомогою еволюційних стратегій [8]. Іншим підходом був метод еволюційного програмування Лоуренса Дж. Фогеля, який був запропонований для створення штучного інтелекту. Еволюційне програмування спочатку використовувало машини кінцевого стану для прогнозування середовищ, а також використовувало варіації і вибір для оптимізації предсказательной логіки [7].

Генетичні алгоритми, зокрема, стали популярними завдяки роботі Джона Холланда на початку 1970-х років і, зокрема, його книзі «Адаптація в природних і штучних системах» (1975). Його робота почалася з вивчення клітинних автоматів, що проводяться Голландією і його учнями в Мічиганському університеті [7]. Голландія представила формалізовану структуру для прогнозування якості наступного покоління, відомого як теорема Голландії за схемою. Дослідження в області генетичних алгоритмів були в основному теоретичними до середини 1980-х років, коли в Піттсбурзі, штат Пенсільванія, відбулася Перша міжнародна конференція по генетичним алгоритмам [8].

Висновки до розділу 1

В даному розділі надається теоретичний матеріал стосовно алгоритмів які використовуються в нейронних мережах. Розглянута історія виникнення штучних нейронних мереж та основні праці, які поклали їм основу. Наведена докладна класифікація нейронних мереж.

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

2.1. Алгоритм навчання лінійних класифікаторів Розенблата та Козинця

Перш ніж описати пропоновану технологію виконання статистичного експерименту нагадаємо основні положення алгоритмів навчання Розенблатта і Козинца на прикладі розпізнавання двох класів V_1 і V_2 [14-16].

Нехай в N - вимірному просторі ознак задана вибірка спостережень з відомою приналежністю до класів

$$X = \{(x_1^{(N)}, c_1), (x_2^{(N)}, c_2), ..., (x_n^{(N)}, c_n)\}, \qquad (2.1)$$

де n - кількість елементів у вибірці, $x_j^{(N)} \stackrel{\Delta}{=} (x_1, ...x_N)$ - точки (N - мірні вектора), $j = 1,...n$ а c_n - індикаторна змінна така, що

$$c_j = \begin{cases} +1, & \text{якщо } x_j^{(N)} \in V_1, \\ -1, & \text{якщо } x_j^{(N)} \in V_2. \end{cases} \quad j = 1,...n \qquad (2.2)$$

Передбачається, що спостереження класів V_1 і V_2 можуть бути розділені лінійною дискримінантною функцією

$$D(x) = \langle w, x \rangle \stackrel{\Delta}{=} \sum_{j=0}^{N} w_j x_j, \qquad (2.3)$$

в якій для зручності запис $\langle w, x \rangle$ позначає скалярний добуток $(N+1)$ - мірних векторів $w = (w_0, w_1, ..., w_N)$ - параметрів (ваг) дискримінантної функції і розширених векторів $x = (1, x_1, ... x_N)$.

Завдання полягає в тому, щоб по кінцевої навчальній вибірці (2.1) з відомими значеннями індикаторної змінної (2.2) визначити вектор $w = (w_0, w_1, ..., w_N)$ параметрів дискримінантної функції (2.3), яка дозволяє повністю розділити спостереження вибірки по схемі:

$$\text{рішення на користь } V_1, \text{ якщо } \langle w, x \rangle > 0, \qquad (2.4)$$

$$\text{рішення на користь } V_2, \text{ якщо } \langle w, x \rangle < 0. \qquad (2.5)$$

Ідея обох алгоритмів полягає в реалізації ітераційних процедур, які дозволяють на основі послідовного перегляду точок навчальної вибірки (2.1) коригувати деяке початкове значення вектора. В результаті такої корекції після певного числа ітерацій дискримінантна функція забезпечить безпомилковий поділ елементів вибірки за схемою (2.4), (2.5).

Різниця алгоритмів навчання полягає в механізмі корекції.

Алгоритм навчання Ф.Розенблатта, запропонований в роботі [14], зводиться до виконання таких кроків:

1. Здається початкове значення вектора ваг $w(0)$ випадковим чином. Наприклад, для двовимірного випадку $(N = 2)$ можна задати $w^{(0)} = (0, 0, 1)$.

2. Послідовно вибираються спостереження $x_\alpha^{(t)} = x^{(N)}$, $\alpha = 1, ..., n$, навчальної вибірки (2.1) і відповідно до (2.3) визначаються значення дискримінантної функції $D(w^{(t-1)}, x_\alpha^{(t)},)$ при поточному значенні вектора $w^{(t-1)}$, $t = 1, 2, ...$.

3. Обчислюється помилка

$$\delta_{\alpha}^{(t)} = \left(D\left(w^{(t-1)}, x_{\alpha}^{(t)}\right) - c(x_{\alpha}^{(t)})\right), \qquad (2.6)$$

що представляє собою різницю значення дискримінантної функції (2.3) і відомого значення індикаторної змінної (2.2), що відповідає обраному спостереженню $x_{\alpha}^{(t)}$.

4. При неправильної класифікації поточного спостереження $x_{\alpha}^{(t)}$ (рис. 2.1, b) вектор ваг модифікується в таким чином:

$$w^{(t)} = w^{(t-1)} + \gamma\left(\delta_{\alpha}^{(t)}\right)^{T} x_{\alpha}^{(t)}, \qquad (2.7)$$

де - заданий темп корекції (рис. 2.1, b).

1. Кроки 2-4 повторюються до тих пір, поки всі точки вибірки (2.1) будуть класифіковані правильно (рис. 2.1, c).

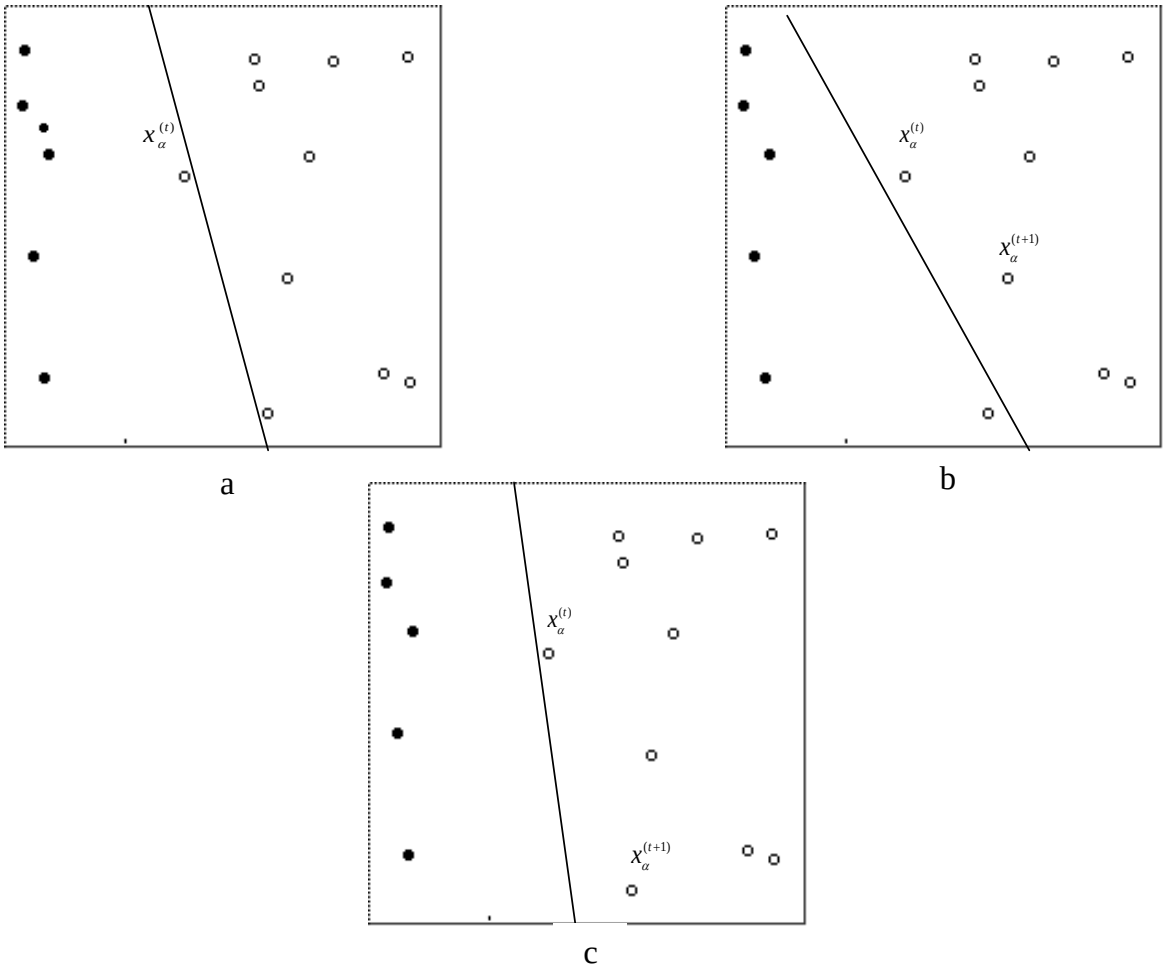


Рисунок 2.1. Графічна інтерпретація алгоритму навчання Розенблата.

На лінійно-роздільній вибірці спостережень описаний процес навчання завершується після закінчення кінцевого числа ітерацій, коли припиняється зміна вектора ваг або сумарна абсолютна помилка по всім спостереженням стає менше деякого малого значення.

Прийняття рішень про приналежність поточного спостереження $(x_1, x_2, ..., x_N)$ класу V_1 чи V_2 реалізується схемою, названою персептроном.

У найпростішому випадку можна розглядати роботу персептрон в двовимірному просторі (при $N = 2$). В цьому випадку персептрон має два входи і дискримінантна функція, яка розділяє двовимірні вектори (точки на площині), що належать класам V_1 і V_2, Являє собою пряму лінію

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0 \qquad (2.7)$$

або, вважаючи $x_0 = -1$ і $w_0 = \theta$, в еквівалентній формі запису

$$w_1 x_1 + w_2 x_2 - \theta = 0. \qquad (2.8)$$

У цьому окремому випадку корекція поточних параметрів двовимірної дискримінантної функції зводиться до додавання коригувальних поправок (рис. 2.1):

$$\begin{aligned} w_{t+1} &= w_t + \gamma(\delta^\alpha)^T x_t, \\ \theta_{t+1} &= \theta_t + \gamma(\delta^\alpha)^T x_t, \end{aligned} \qquad (2.9)$$

де $\gamma = \text{const}$ - швидкість навчання.

В роботі [15, 17] доведена теорема, відповідно до якої для лінійно роздільної вибірки кінцевого числа спостережень існує таке число ітерацій

$$t^0 \le \frac{D^2}{\varepsilon^2}, \qquad (2.10)$$

при якому вектор θ_{t^0} , Задовольняє нерівність $\left\langle \theta_{t^0}, x_j^{(N)} \right\rangle > 0$ для всіх $j \in (1,n)$, причому

$$D = \max_{j \in (1,n)} \left| x_j^{(N)} \right|,$$

$$\varepsilon = \min_{x \in \mathrm{Co}(X)} \left| x^{(N)} \right| > 0,$$

де $\mathrm{Co}(X)$ - опукла оболонка множини X .

В роботі [15] Б.М. Козинець запропонував інший алгоритм навчання, який отримав назву алгоритм Козинця.

Основна ідея алгоритму полягає в тому, що на кожному кроці ітерації $t = 1,2,...$ відбувається пошук такого спостереження $x_\alpha^{(t)} = x^{(N)}$, $\alpha = 1,...,n$, навчальної вибірки (2.1), яке неправильно класифікується при поточному значенні вектора параметрів $w^{(t-1)}$ дискримінантної розділяючої функції. Якщо таких векторів немає для всіх точок навчальної вибірки, то алгоритм завершує свою роботу.

Якщо ж знайдений вектор $x_\alpha^{(t)}$, який неправильно класифікується, то проводиться корекція вектора параметрів наступним чином [11, с. 187]:

$$w^{(t)} = (1 - \gamma^{(t)}) \cdot w^{(t-1)} + \gamma^{(t)} \cdot x_\alpha^{(t)}, \qquad (2.11)$$

де

$$\gamma = \arg\min \left| (1 - \gamma^{(t)}) \cdot w^{(t-1)} + \gamma^{(t)} \cdot x_\alpha^{(t)} \right|. \qquad (2.12)$$

Дамо графічну інтерпретацію алгоритму Козинца для випадку $N = 2$, коли за допомогою прямої лінії потрібно розділити дві множини точок на площині. Алгоритм зводиться до виконання таких кроків:

1. Випадковим чином вибираються дві точки навчальної вибірки, що належать різним класам, між яким проводиться пряма AB (рис. 2.2, a).

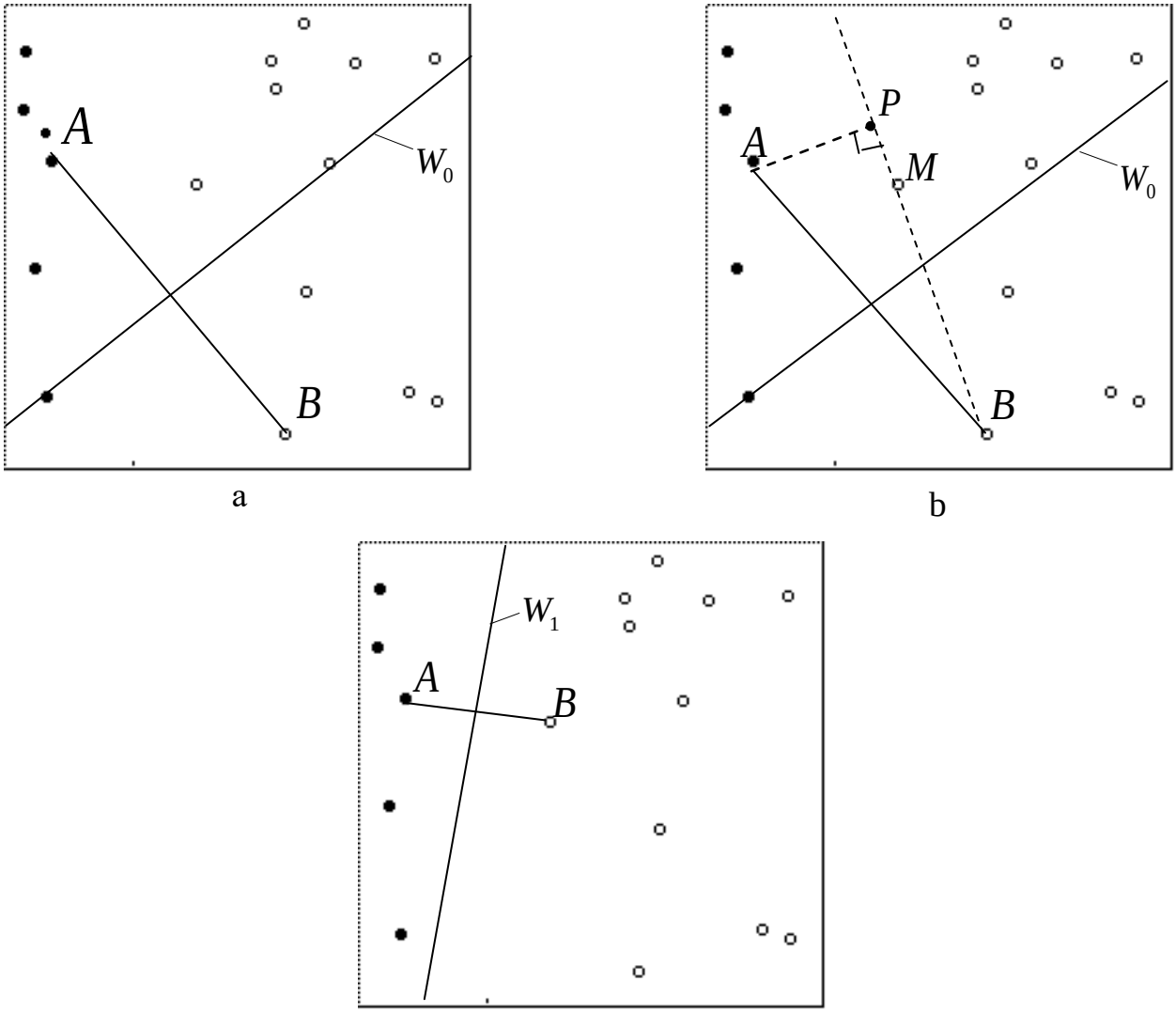


Рисунок 2.2. Графічна інтерпретація алгоритму Козинця на площині (випадок 1).

2. Параметри серединного перпендикуляра W_0 до відрізка AB визначають початкове наближення вектора параметрів $w^{(0)}$ шуканої дискримінантної функції.

3. Вибирається довільна точка M і по знаку індикаторної змінної (2.2) визначається її приналежність до одного з класів. Точка M з'єднується прямою з точкою того ж класу, а з точки A протилежного класу опускається перпендикуляр на зазначену пряму (рис. 2.2, b).

4, a. Якщо основа перпендикуляра AP виходить за межі прямої BM (рис. 2.2, b), то точка M визначає нове положення прямої AB (рис. 2.2, c), за допомогою якої знаходять параметри $w^{(1)}$ скоригованої дискримінантної функції W_1.

4, b. Якщо ж підстава перпендикуляра AP лежить в межах відрізка BM (рис. 2.3, b), то нове положення прямої AB (а значить і параметри скоригованої дискримінатної функції W_1) визначає вже точка P.

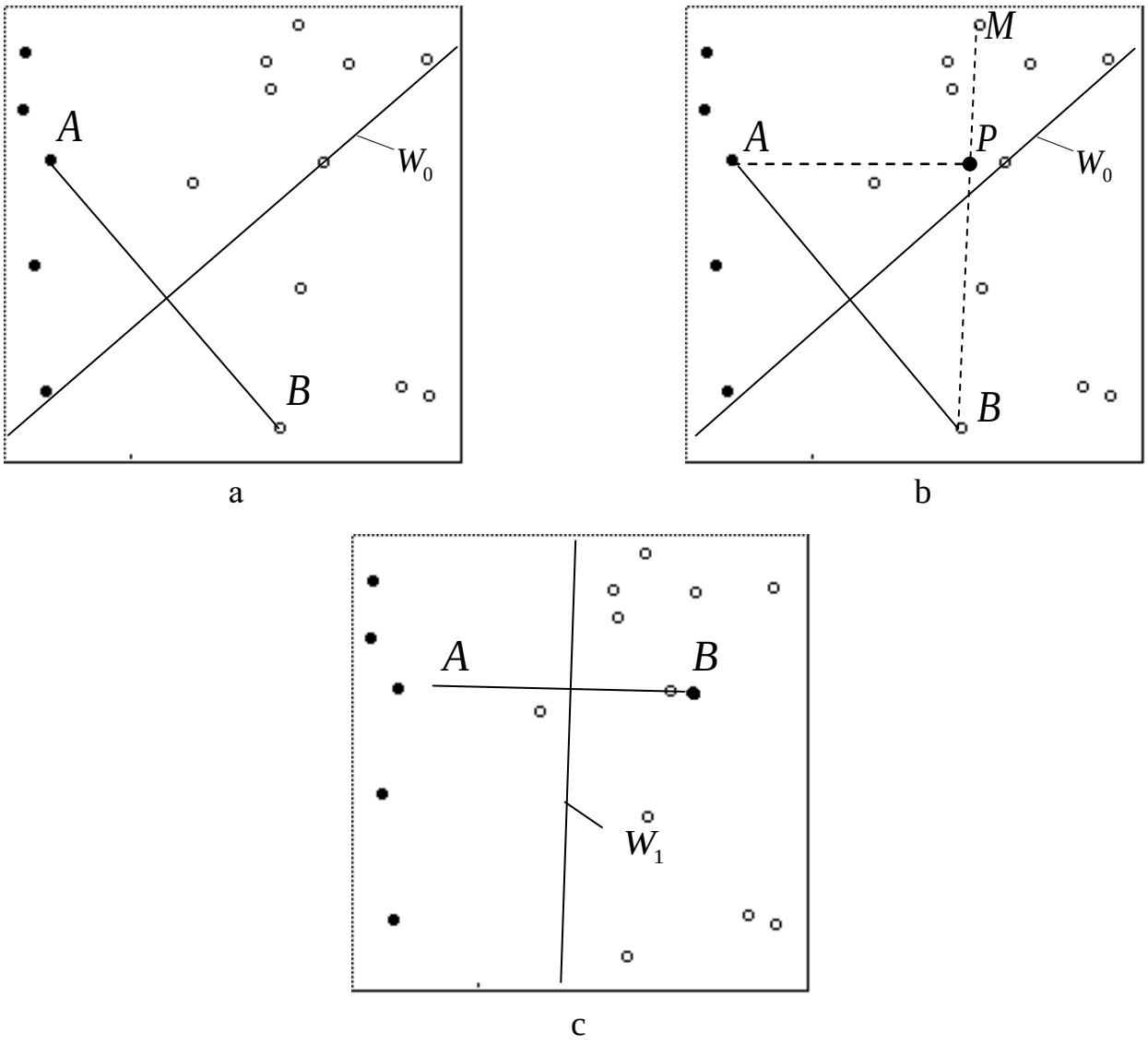


Рисунок 2.3. Графічна інтерпретація алгоритму Козинця на площині (випадок 2).

Кроки 1-4 повторюються до тих пір, поки всі крапки навчальної вибірки (2.1) будуть правильно класифіковані.

Існує теорема [15, 17], відповідно до якої алгоритм Козинця сходиться за ко-кінцевих число ітерацій

$$t_0 \le \frac{Q^2}{\varepsilon^2} \ln \frac{Q^2}{\varepsilon^2}. \qquad (2.13)$$

На перший погляд з порівняння оцінок (2.8) і (2.13) може здатися, що алгоритм Розенблатта завжди сходиться при меншій кількості ітерацій. Однак, як зазначено в [11, с.167] оцінки (2.8) і (2.13) досить грубі, а, значить, висновок про перевагу алгоритму Розенблатта на підставі цих оцінок не правомірний [48]. Тому пропонується провести порівняння швидкості збіжності алгоритмів на основі статистичного експерименту.

2.2. Статистичний експеримент та метод Монте-Карло

Статистичним експеримент являє собою ефективний метод дослідження поведінки деякого об'єкта (моделі, алгоритму) під впливом вхідних впливів, які носять випадковий характер. В результаті такого дослідження можуть бути визначені характеристики системи, аналітична оцінка яких скрутна або неможлива[17].

Статистичний експеримент найчастіше використовує метод Монте-Карло [17], в основі якого лежить генерація випадкових (псевдовипадкових) чисел з заданим розподілом ймовірностей [18]. При цьому замість вирішення аналітичної задачі можна моделювати випадковий процес і обчислювати його статистичні оцінки (ймовірність, математичне сподівання, дисперсію) для наближеного рішення аналітичної задачі [19, 20].

При статистичному експерименті проводиться оцінка поведінки досліджуваного об'єкта при багаторазових випробуваннях з різними конкретними значеннями випадкових чинників з подальшою статистичною обробкою результатів спостережень [48].

Дотримуючись методології статистичного експерименту, розроблена програмна інструментальна система, що дозволяє проводити експерименти по оцінці швидкості збіжності лінійних класифікаторів алгоритмів Розенблатта і Козинця. Такі експерименти проводилися на вибірках даних, що генеруються методом Монте-Карло [48].

2.3. Програмні засоби, які були використані для реалізації

Для реалізації поставлених задач було використано наступні програмні засоби Microsoft Visual Studio 2017 (мова програмування – C#), що являє собою систему для розробки програмного засобу.

Microsoft Visual Studio надає розвинений редактор коду, вбудований відладчик та інші засоби, що спрощують розробку додатків на мові C#. Також має наступні переваги:

1. CodeLens - являє із себе підказки, які з'являються над вашим кодом, що надають інформацію про те, які залежно є у цього коду, результати тестів цього методу, хто міняв цей код, пов'язані робочі елементи;
2. IntelliTrace - Можливості IntelliTrace значно підвищують продуктивність налагодження. IntelliTrace автоматично веде журнали виконання коду, запам'ятовує і відзначає події в таймлайнах, які далі можна переглядати, переміщатися і перевіряти стан;
3. CodeMap - Ця можливість стане в нагоді при роботі з великими кодовими базами. Підтримується також переміщення по створеній карті з паралельно відкритим кодом. Це допомагає відслідковувати ваше місцезнаходження в коді під час роботи;

4. Доступні інструменти для побудови процесів управління проектами та командною роботою: Team Foundation Server або Visual Studio Online Advanced;

5. Можливість підключення бібліотек з відкритим вихідним кодом.

C#:

1. Стандартизований. Перша версія C # стандартизована в ECMA (Standard ECMA-334 C # Language Specification, 3rd edition (June 2005)) та ISO (ISO / IEC 23270: 2003, Information technology - C # Language Specification);

2. JIT-компіляція;

3. Можливість використання стандартних бібліотек програмування включених в .NET Framework, що включає широкий спектр API-інтерфейсів для роботи як з самим обладнанням так і з операційною системою;

4. Компіляція в CIL (common intermediate language) код, який може виконуватися на будь-якій машині, де підтримується CLR (common language runtime);

5. CodeLens доступний тільки для C#;

6. Garbage collector.

2.4. Проектування програмного продукту

2.4.1. Модель життєвого циклу

Модель життєвого циклу - структура, що складається із процесів, робіт та задач, які включають в себе розробку, експлуатацію і супровід програмного продукту; охоплює життя системи від визначення вимог до неї до припинення її використання [27].

Каскадна модель життєвого циклу (модель водоспаду, англ. waterfall model) у 1970 р. була запропонована У. Ройсом. Принциповою

особливістю каскадної моделі є т, що перехід на кожну наступну стадію здійснюється тільки після повного завершення роботи поточної стадії, повернення на попередні стадії не передбачено. Кожна з стадій закінчується одержанням результатів, що є вхідними даними для наступної стадії, та випуском повного комплекту документації. Вимоги до ПЗ, визначені на стадії формування вимог, документуються у вигляді технічного завдання і фіксуються на весь час розроблення. Критерієм якості розробки за такої моделі є точність виконання специфікацій технічного завдання [27].

Цінність цієї моделі полягає в тому, що вона фіксує послідовність етапів розроблень та можливість повернення до попередніх етапів роботи.

Основна увага розробників має зосереджуватися на досягненні найкращих значень технічних характеристик ПЗ, а саме: продуктивності, обсягу пам'яті тощо [27].

Каскадна стратегія (одноразовий прохід, водоспадна або класична модель) має на увазі лінійну послідовність виконання стадій створення інформаційної системи (рис. 2.4). Іншими словами, перехід з однієї стадії на наступну відбувається тільки після того, як буде повністю завершена робота на поточному [27].

Дана модель застосовується при розробці інформаційних систем, для яких на самому початку розробки можна досить точно і повно сформулювати всі вимоги [27].

Переваги моделі:

- на кожній стадії формується закінчений набір документації, програмного і апаратного забезпечення, який відповідає критеріям повноти і узгодженості;
- виконуються в чіткій послідовності стадії дозволяють впевнено планувати терміни виконання робіт і відповідні ресурси (грошові, матеріальні і людські) [27].

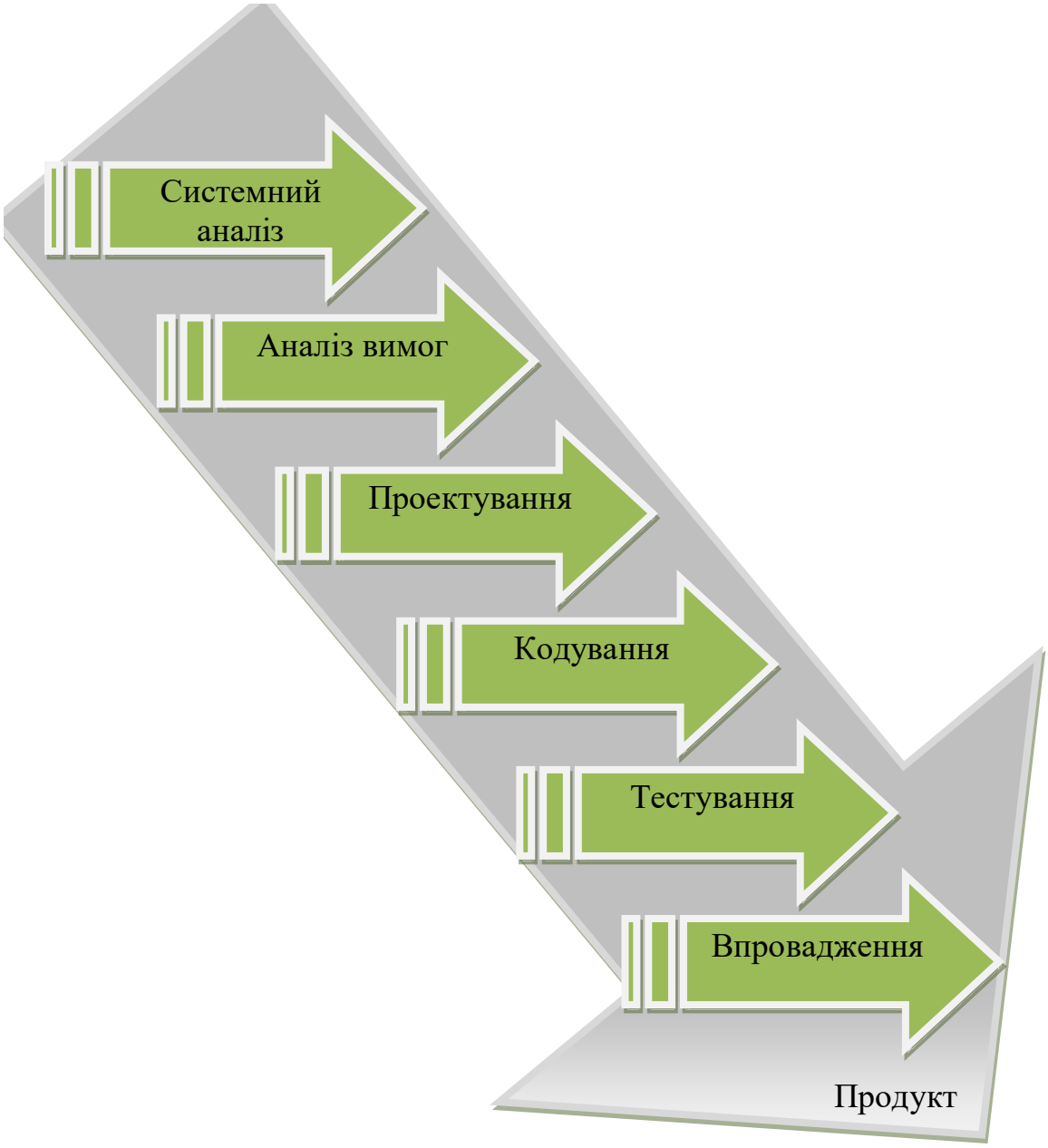


Рисунок 2.4. Каскадна стратегія.

Недоліки моделі:

- реальний процес розробки інформаційної системи не часто повністю відтворює таку не гнучку схему. Особливо це має відношення до розробки не типових і новаторських систем;
- заснована на точному формулюванні всієї групи вихідних вимог до інформаційної системи. В реальності ж на початку проекту вимоги часто визначено не повністю;

- основним недоліком є те, що результат розробки доступний замовнику тільки в кінці проекту. У випадку неточного встановлення вимог або їх частої зміни протягом періоду створення ПП замовник отримає продукт, який не відповідає його потребам [27].

Інкрементна стратегія (англ. Increment - збільшення, прирощення) має на увазі розробку інформаційної системи з стадіями в лінійній ослідовності, але з декількома інкрементами (версіями), тобто з запланованими поліпшеннями продукту (рис. 2.5) [27].

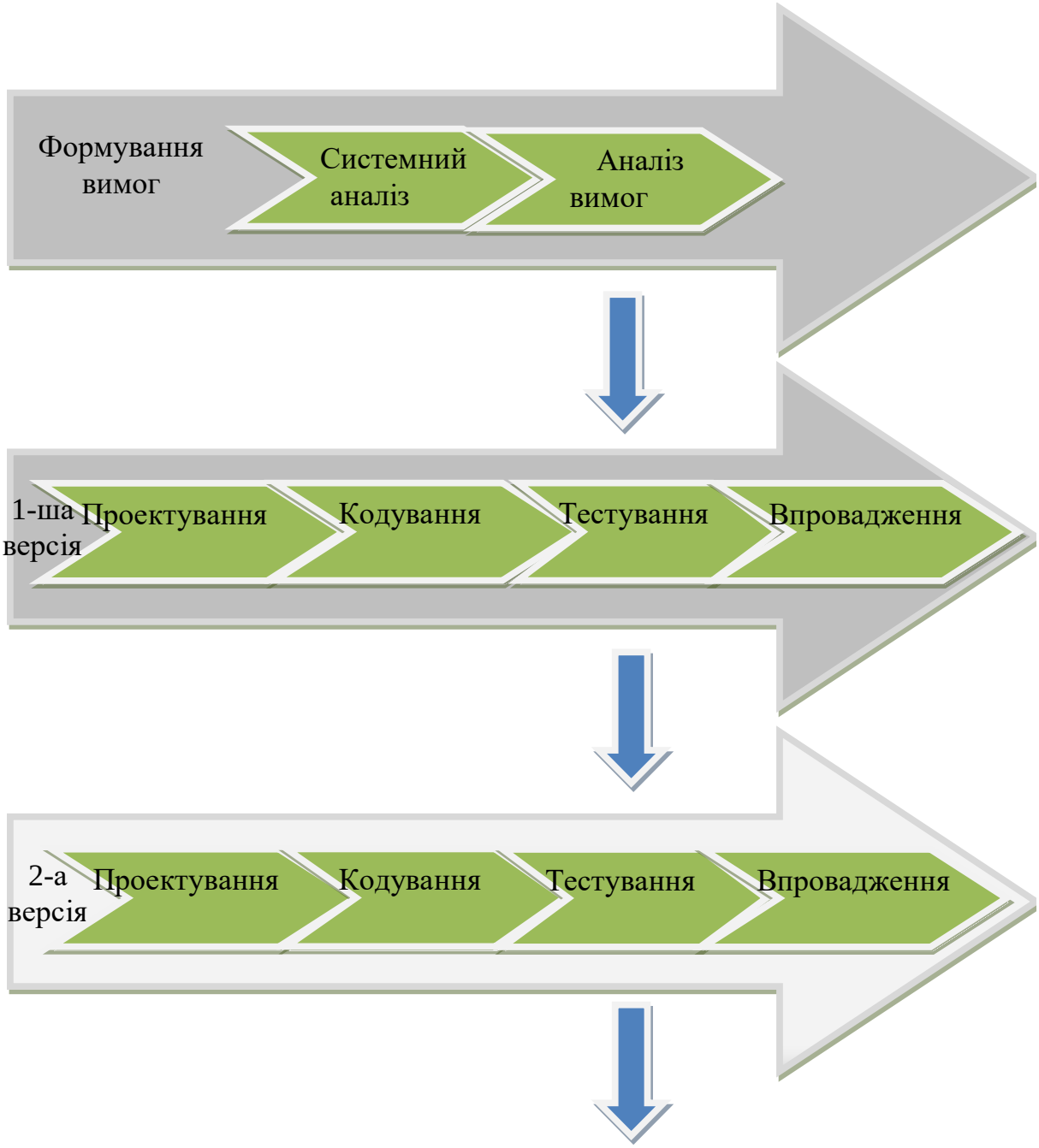


Рисунок 2.5. Інкрементна стратегія.

На початку роботи з проектами маю бути визначені всі основні вимоги замовника до системи, далі виконується власне її розробка як послідовності версій. В той же час, кожна з версій є закінченою і являє собою працездатний продукт. Перша з цих версій реалізує лише частину з можливостей, які були заплановано, кожна наступна інкремента реалізує якісь з додаткових можливості і т. д., до поки не буде отримано готовий програмний продукт [27].

Ця модель життєвого циклу продукту характерна для розробки складної і комплексної системи, для якої наявні чіткі вимоги (як з боку замовника, так і з боку розробника) до того, яким має бути врешті решт кінцевий продукт (інформаційна система) [27]. Розробка версіями через безліч різних причин:

- відсутність у замовника можливості фінансування всього довгострокового проекту відразу;
- відсутність у розробника необхідних ресурсів для реалізації складного проекту в стислі терміни;
- через наявність вимог до поетапного впровадження і освоєння продукту кінцевим користувачем. Впровадження всієї системи відразу може викликати у її користувача відразу і тільки «загальмувавши» процес переходу на нові технології можливо сподобатися [27].

Переваги і недоліки цієї стратегії такі ж, як і у класичної. Але на відміну від класичної стратегії замовник може раніше побачити результати. Уже за результатами розробки та впровадження першої версії він може незначно змінити вимоги до розробки, відмовитися від неї або запропонувати розробку більш досконалого продукту з укладенням нового договору [27].

Спіральна стратегія (еволюційна або ітераційна модель, автор Баррі Боем, 1988 г.) має на увазі розробку у вигляді послідовності версій, але на початку проекту визначені не всі вимоги (рис. 2.6). Вимоги уточнюються в результаті розробки версій [27].

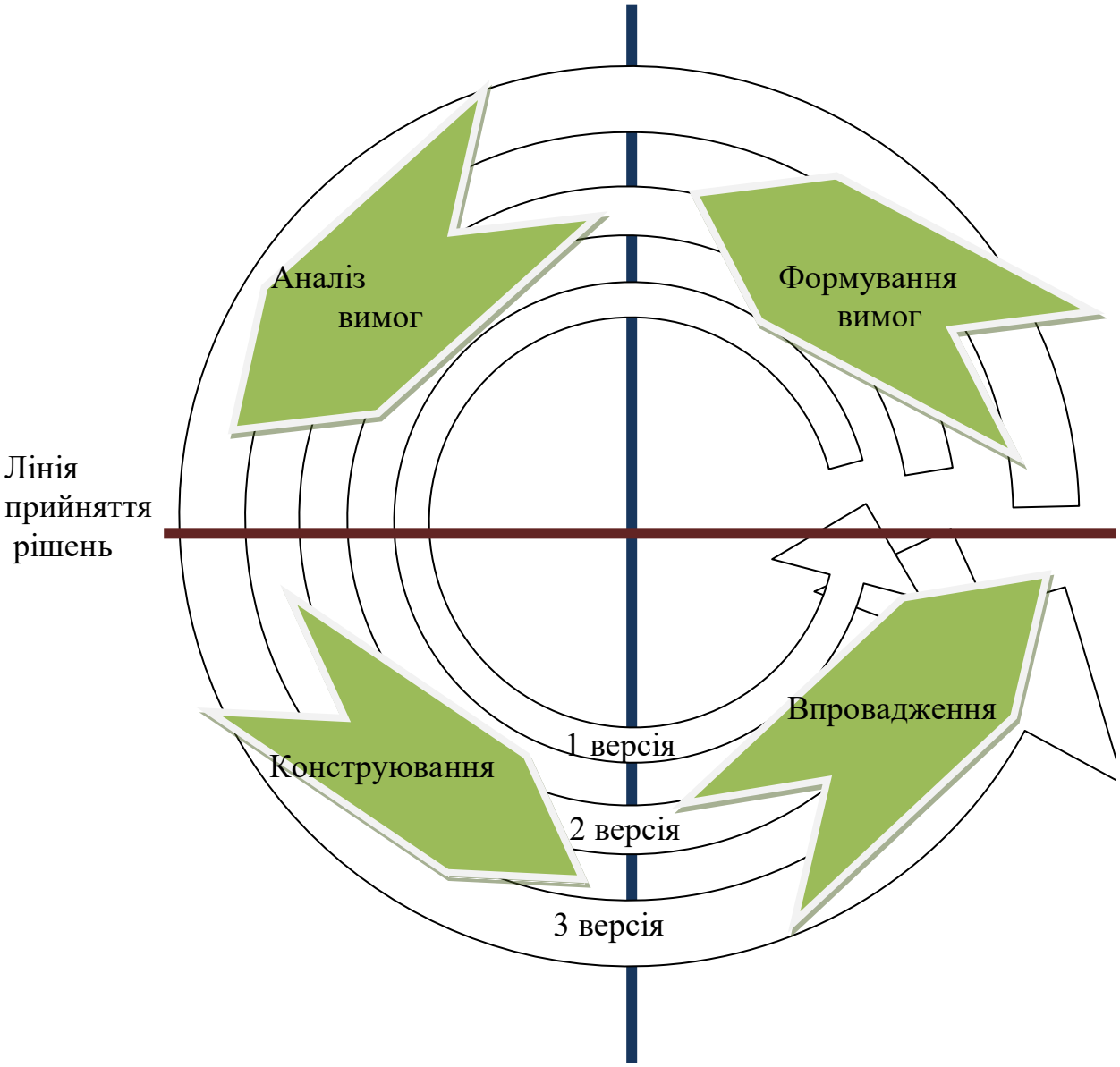


Рисунок 2.6. Спіральна стратегія.

Дана модель життєвого циклу характерна при розробці новаторських (нетипових) систем. На початку роботи над проектом у замовника і розробника відсутнє чітке бачення кінцевого продукту (вимоги не можуть бути чітко визначені) або стовідсоткової впевненості успішної реалізації проекту (ризики дуже великі). У зв'язку з цим приймається рішення розробки системи по частинах з можливістю зміни вимог або відмови від її подальшого розвитку. Як видно з рис.2.6, розвиток проекту може бути завершено не тільки після стадії впровадження, але і після стадії аналізу ризику [27].

Переваги моделі:

- швидше показує користувачам системи працездатні готові продукти, таким чином, активізує процеси щодо уточнення і доповнення вимог;
- має можливість зміни вимог при розробці інформаційної системи, що є характерним для більшості розробок;
- більша гнучкість відносно управління проектом;
- дозволяє отримати більш надійну і стійку систему. В процесі розвитку системи помилки і слабкі місця виявляються і можуть бути виправлені на будь-якій з ітерацій;
- дозволяє удосконалювати процес розробки - аналіз, проведений в кожній ітерації, дозволяє проводити оцінку того, що повинно бути змінено в організації розробки, і поліпшити її на наступній ітерації;
- зменшуються ризики замовника. Замовник може з мінімальними для себе фінансовими втратами завершити розвиток безперспективного проекту [27].

Недоліки моделі:

- збільшується невизначеність у розробника в перспективах розвитку проекту. Цей недолік випливає з попереднього гідності моделі;
- ускладнення операцій тимчасового і ресурсного планування всього проекту в цілому. Для знаходження виходу з цієї ситуації необхідно ввести тимчасові обмеження на кожну зі стадій життєвого циклу. Навіть якщо не вся запланована робота виконана, перехід все рівно здійснюється відповідно до плану. План складається на основі статистичних даних, отриманих в попередніх проектах і особистого досвіду розробників [27].

Знання різних моделей життєвого циклу і вміння їх застосовувати на практиці необхідні будь-якому керівникові проекту. Правильний вибір моделі дозволяє грамотно планувати обсяги фінансування, терміни і ресурси, необхідні для виконання робіт, скоротити ризики як розробника, так і замовника. Це сприяє підвищенню авторитету (іміджу) розробників в

очах замовника і в свою чергу впливає на перспективу подальшої співпраці з ним та іншими замовниками. Вважати, що спіральна модель краще за інших, невірно. Адже на кожен проект полягає окремий договір з певною вартістю. Укладати договір на велику суму з невизначеним підсумковим результатом замовник ніколи не буде (якщо тільки він не альтруїст). У цьому випадку він запропонує вкласти спочатку невелику суму в проект і вже за результатами першої версії (ітерації) буде вирішувати питання про укладення додаткового договору на розвиток системи [27].

Кожна з моделей має свої переваги і недоліки, а також сфери застосування в залежності від специфіки розробляється, можливостей замовника і розробника і т. п. У табл. 2.1 наводиться порівняльна характеристика розглянутих вище моделей, яка повинна допомогти у виборі стратегії для конкретного проекту [27].

Таблиця 2.1

Порівняння моделей життєвого циклу

<table>
<tr><td rowspan="2">Характеристика проекту</td><td colspan="3">Модель (стратегія)</td></tr>
<tr><td>Каскадна</td><td>Інкрементна</td><td>Спіральна</td></tr>
<tr><td rowspan="2">Новизна розробки та забезпеченість ресурсами</td><td colspan="2">Типовий. Добре опрацьовані технологія і методи вирішення завдання</td><td>Нетипової (новаторський). нетрадиційний для розробника</td></tr>
<tr><td>Ресурсів замовника і розробника вистачає для реалізації проекту в стислі терміни</td><td colspan="2">Ресурсів замовника або розробника НЕ вистачає для реалізації проекту в стислі терміни</td></tr>
</table>

Продовж. табл. 2.1

Характеристика проекту	Модель (стратегія)		
	Каскадна	Інкрементна	Спіральна
Масштаб проекту	Малі та середні проекти	Середні і великі проекти	Будь-які проекти
Терміни виконання	До року	До декількох років.	
Висновок окремих договорів на окремі версії	Укладається один договір. Версія і є підсумковий результат проекту	На окрему версію або кілька послідовних версій зазвичай укладається окремий договір	
Визначення основних вимог на початку проекту	Так	Так	Немає
Розробка ітераціями (версіями)	Немає	Так	Так
Поширення проміжного програмного забезпечення	Немає	Може бути	Так

У табл. 2.1 не варто вважати «Так» і «Ні» жорсткими вимогами. Під час використання каскадної моделі (наприклад, додавання деяких непередбачених раніше розробником чи замовником функцій) незначна зміна вимог у міру розвитку проекту зустрічається не так уже й рідко і в

разі їх реалізації сприяє поліпшенню взаємин між сторонами. Аналогічно поширення проміжного програмного забезпечення при спіральної моделі необов'язково, а іноді навіть шкідливо відбивається на процесах впровадження і дослідної експлуатації системи [27].

При розробці системи під підсумковим продуктом і проміжним програмним забезпеченням згідно слід розуміти, що:

- ревізія (виправна або досвідчена) - будь-які оперативні зміни програмного та інформаційного забезпечення, а також БД, необов'язкові в даний момент до передачі на об'єкти впровадження та пов'язані з усуненням помилок і удосконаленням;
- модифікація - будь-які оперативні зміни програмного та інформаційного забезпечення, а також БД, обов'язкові для передачі на об'єкти впровадження і обумовлюють зміну експлуатаційних характеристик без зміни функцій (передбачених ТЗ), а також зміни, пов'язані з усуненням помилок, удосконаленням;
- версія - будь-які зміни програмного та інформаційного забезпечення, а також БД, обов'язкові для передачі на об'єкти впровадження, які дозволяють виконувати заявлені або додаткові функції, а також забезпечують перехід на нові операційні системи і інформаційне середовище;
- розвиток (черга) - планові зміни інформаційної системи, пов'язані з введенням нових функцій і поліпшенням експлуатаційних характеристик, переходом на нову інформаційну середу, впровадженням нових комплексів технічних засобів, нових інформаційних технологій та ін [27].

Поміж існуючих моделей життєвого циклу (каскадна, гнучка, спіральна та еволюційна) було обрано каскадну модель. Оскільки це поетапне виконання визначених дій. Також в цій моделі значна увага приділяється інженерії вимог та власне проектуванню, що застраховує від вагомих помилок [27].

В даний час є кілька методологій розробки програмного забезпечення, які можна рекомендувати при використанні каскадної моделі життєвого циклу. Найбільш відомими з них є методологія швидкої розробки додатків (Rapid Application Development, RAD) і екстремальне програмування (eXtreme Programming, XP - автор Кент Бек, 1999) [27].

Методологія RAD. На початковому етапі існування комп'ютерних інформаційних систем їх розробка велася на традиційних мовах програмування і мала на увазі, як правило, ручний набір текстів програм. Однак у міру зростання складності розроблюваних систем і збільшення запитів користувачів потрібні були нові засоби, щоб забезпечити значне скорочення термінів розробки. Це стало передумовою для створення інструментального засобу для швидкої розробки додатків. Розвиток таких напрямків призвело до того, що на ринку з’явилися розробки програмного забезпечення засобів автоматизації практично всіх стадій життєвого циклу [27].

Під RAD-розробкою зазвичай розуміють процес розробки, який складається з трьох елементів:

- невеликої команди з програмістів (до десяти осіб);
- короткого, але ретельно пропрацьованого виробничого графіку (від 2 до 6 місяців);
- повторюваного циклу, під час якого розробник коли додаток починає набувати форми, реалізує в продукті нові вимоги, отримані від замовника [27].

Крім особливостей, які характеризують спіральну модель життєвого циклу програмної системи, методологія RAD предбачає використання на кожній з ітерацій:

- CASE 2 - засобів формування та аналізу вимог, проектування системи, автоматичної генерації коду програм і структури БД, а також автоматичного тестування програмного забезпечення;

- інструментальних засобів, що забезпечують візуальну розробку (програмування) системи. Середовище розробки додатків дозволяє без написання коду програми створювати («малювати») складні графічні інтерфейси користувача, склад і структуру БД, запити до БД, а також пов'язувати дані з елементами інтерфейсу (перемикачами, полями введення, таблицями і т. Д.);

- інструментальних засобів, що підтримують об'єктно-орієнтований підхід. Ці засоби дозволяють створити опис предметної області у вигляді сукупності об'єктів - сутностей реального світу, характеризуються властивостями (атрибутами) і поведінкою (методами);

- інструментальних засобів, що забезпечують поділ програмування. Кожен об'єкт, що входить до складу програми, може генерувати події і реагувати на події, що генеруються іншими об'єктами;

- шаблонів і бібліотек готових рішень як власної розробки, так і сторонніх виробників [27].

Умови застосування методології RAD:

- застосовна для відносно невеликих проектів, що розробляються під конкретного замовника;

- непридатна для побудови складних розрахункових програм, операційних систем або програм управління космічними кораблями, тобто програм, що вимагають написання великого обсягу (сотні тисяч рядків) унікального коду;

- непридатна для розробки додатків, в яких відсутня яскраво виражена інтерфейсна частина, наочно визначає логіку роботи системи (наприклад, додатки реального часу);

- непридатна для розробки додатків, від яких залежить безпека людей (наприклад, керування літаком або атомною електростанцією), так як ітеративний підхід припускає, що перші кілька версій, швидше за все не будуть повністю працездатні, що в даному випадку виключається [27].

Методологія XP [27]. Даний підхід орієнтований на розробку інформаційних систем групами малого і середнього розміру в умовах невизначених або швидко змінюються вимог.

Відмінними рисами XP-розробки є:

- часта зміна версій і модифікацій (тривалість ітерацій аж до годин і хвилин, а зазвичай - 2 тижні; в RAD - мінімум 2 місяці);
- безперервний зв'язок із замовником - в групі весь час знаходиться кваліфікований представник замовника [27]. Це найпотаємніше бажання будь-якого розробника. Як правило, дуже важко переконати замовника виділити такого представника, щоб він цілими днями (тижнями) не виконував своїх прямих обов'язків на роботі. Але ще важче буває знайти саме кваліфікованого фахівця, який може оперативно відповісти на всі збільшує кількість питань з боку команди розробників;
- просте проектування - при розробці завжди вибирається найбільш просте рішення [28]. «Краще зробити просту річ сьогодні і завтра заплатити ще трохи за внесення невеликих змін, ніж робити сьогодні складну річ, яка завтра може не знадобитися». Спірне твердження, на яке можна заперечити, що «скупий платить двічі». Нерідко досвідченому розробнику, особливо непогано розбирається в поставленому завданню, видно, що якщо застосувати більш складну схему реалізації деякої функції або розширити структуру даних, то це збільшить коло вирішуваних завдань, дозволить з меншими витратами адаптувати систему під мінливі або нові вимоги замовника і т.д.;
- простий дизайн - система повинна бути спроектована настільки просто, наскільки це можливо на кожен момент часу. Чим інтерфейс простіше, тим швидше і якісніше йде освоєння системи користувачами [27]. Це не означає, що інтерфейс «командного рядка» самий кращий. Якраз навпаки, «дружній» і простий інтерфейс повинен бути інтуїтивно-зрозумілим для користувача; в ньому мають бути відсутні непотрібні елементи, основна мета яких - зробити візуальне враження на користувача;

додаткові діалогові вікна з введення даних в рядок таблиці, коли це можна зробити безпосередньо в рядку цієї таблиці і т. д .;

- колективне володіння кодом - будь-який, хто бачить можливість поліпшити якусь частину коду, може зробити це в будь-який момент часу [27]. Це має на увазі застосування однакових стандартів і правил оформлення коду з вичерпними коментарями, а також і ведення загальнодоступною історії розвитку системи;

- програмування в парах - на пару програмістів доводиться один комп'ютер. Поки один з них безпосередньо програмує, інший обмірковує питання реалізації вимог (функцій, БД і т.п.) [27]. Сенс положення полягає не в економії на матеріально-технічному забезпеченні команди розробників, а в більш розумному поєднанні різних видів діяльності кожного з них. Як показує досвід, при безпосередньому програмуванні, виправленні помилок і т.п. програміст починає думати односторонньо і все більш вузькими категоріями. Саме тому під час «відпочинку від комп'ютера» приходять найвдаліші рішення;

- безперервне і перехресний проектування, розробка, інтеграція і тестування системи [27].

2.4.2. Побудова ієрархічної структури та розрахунок нев’язки

Розрахунок нев’язки здійснювався за допомогою MATLAB (рис. 2.7)

```
>> n = 8; %количество вершин полученного графа
E = 8; %количество ребер полученного графа
Et = n-1; %количество ребер дерева
Ec = n*(n-1)/2; %количество ребер полного графа
Nev = (E-Et)/(Ec-Et); %расчет невязки
Nev %вывод значений невязки

Nev =

    0.0476
```

Рисунок 2.7. Розрахунок нев’язки.

Побудова ієрархічної структури програмного продукту на рис. 2.8.

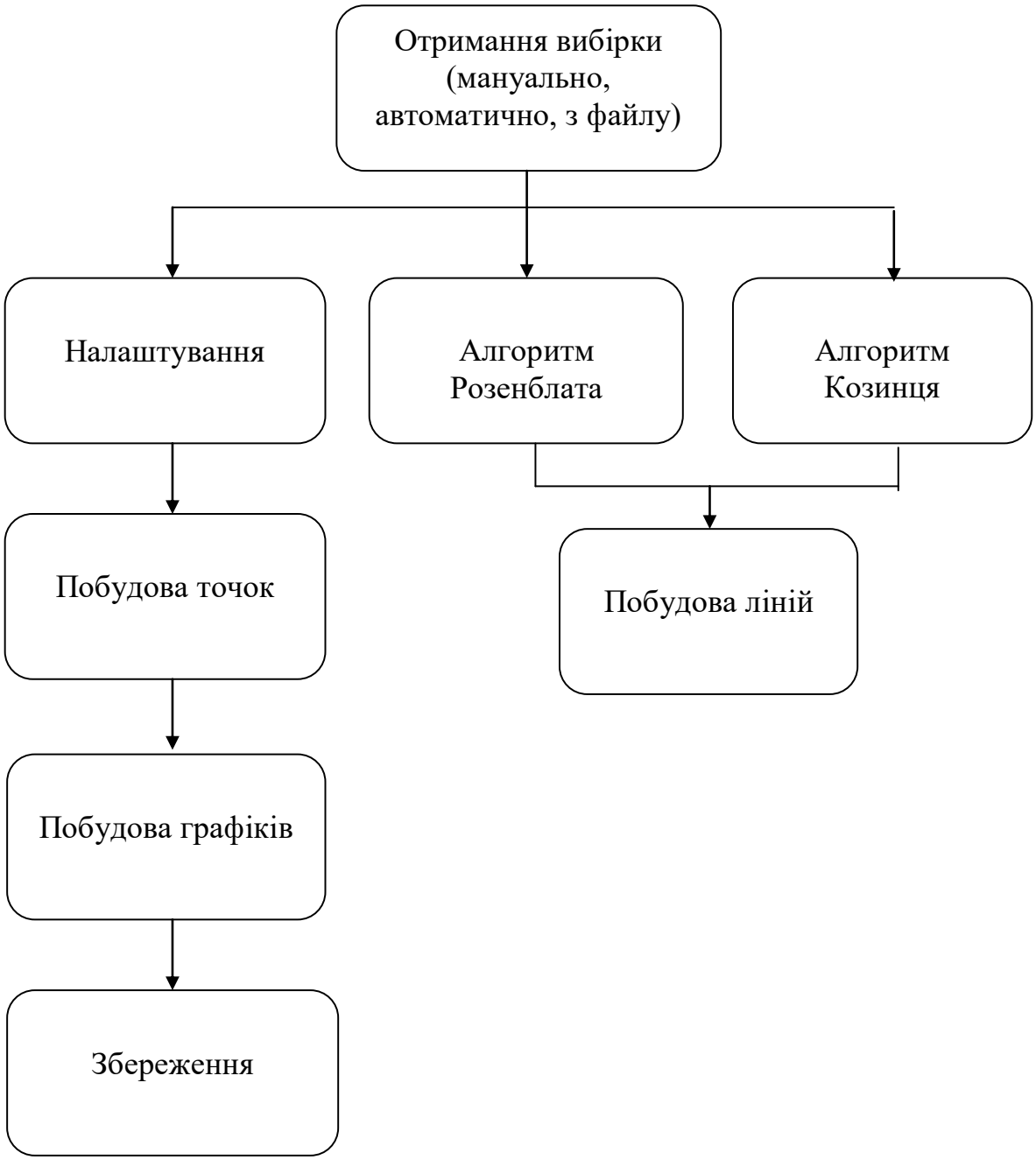


Рисунок 2.8. Ієрархічна структура ПП.

Як можна побачити, продукт отримав дуже малу нев’язку, яка прагне до нуля. Це свідчить про добре сплановану систему, яку доцільно розробляти.

2.4.3. Діаграма сутність-зв’язок (ERD)

Діаграма сутність-зв’язок, яка побудована згідно до завдань, що були сформовані на етапі системного аналізу та аналізу вимог, показана на рис. 2.9 [42].

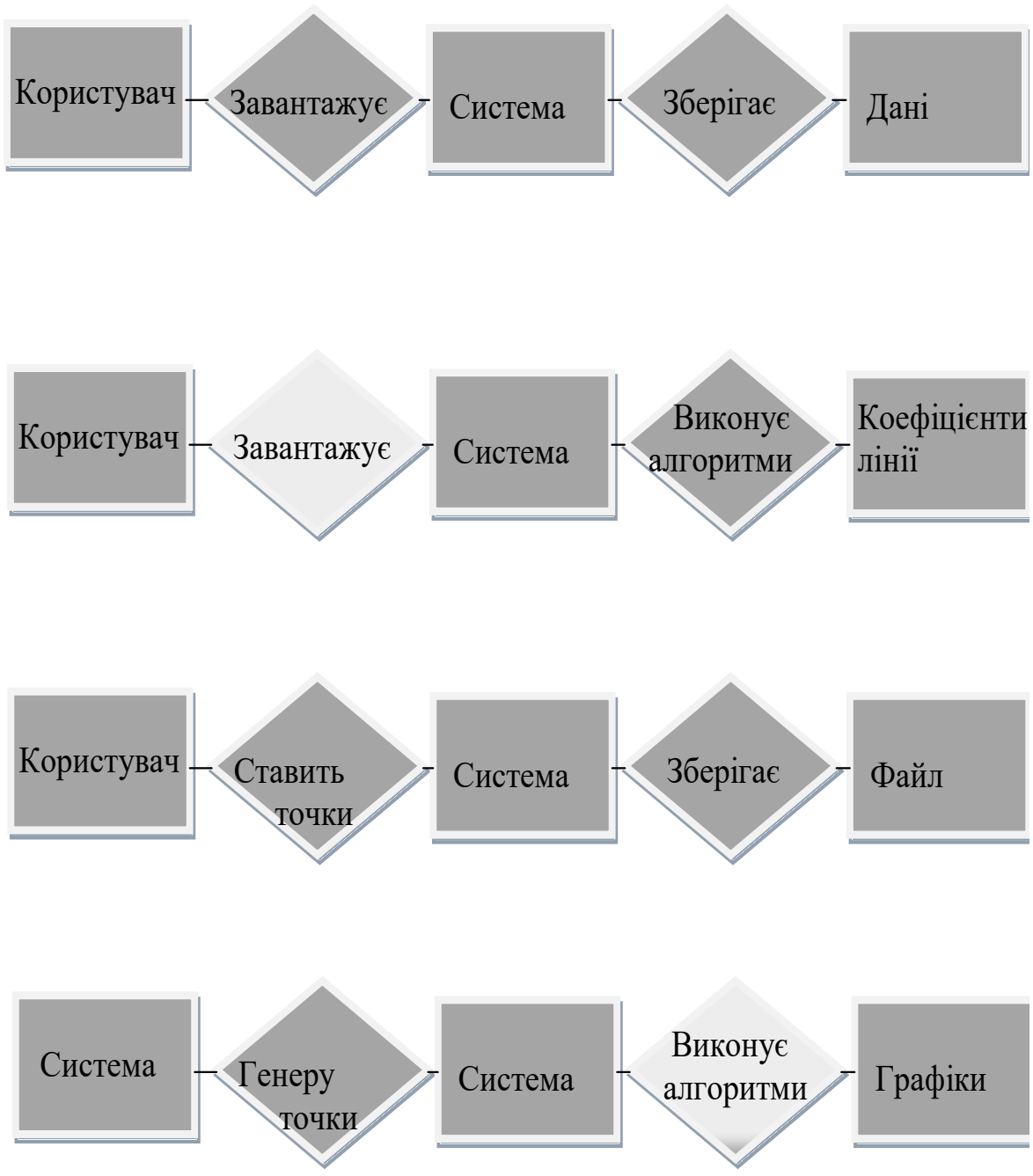


Рисунок 2.9. Діаграма сутність-зв’язок.

2.4.4. Контекстна діаграма IDEF0

Контекстна діаграма IDEF0 використовується для документування процесів виробництва продуктів і відображення інформації про

використання ресурсів на кожному з етапів проектування систем. Вона належить до функціонального моделювання.

На рис. 2.10 зображена контекстна діаграма, що відображає використання всіх вхідних ресурсів проекту. Діяльністю виступає власне створення програмного продукту (ПП). Управлінням виступають наступні ГОСТи [43]:

Таблиця 2.2

Використані ГОСТи

Позначення	Найменування
Стандарти ISO/IEC (ІСО/МЕК) в області розробки і документування програмних засобів	
ГОСТ Р ІСО/МЕК 12207-02	Інформаційна технологія. Процеси життєвого циклу програмних засобів
ГОСТ Р ІСО/МЕК 9126-93	Інформаційна технологія. Оцінка програмної продукції. Характеристики якості і керівництва щодо їх застосування
ГОСТ Р ІСО/МЕК 12119-94	Інформаційна технологія. Пакети програм. Вимоги до якості і тестування
Комплекс нормативних документів на автоматизовані системи	
ГОСТ 34.601-90	Автоматизовані системи. Стадії створення
ГОСТ 34.602-89	Технічне завдання на створення автоматизованої системи
Комплекс стандартів Єдиної системи програмної документації (ЄСПД)	
ГОСТ 19.201-78	Технічне завдання. Вимоги до змісту та оформлення
ГОСТ 19.701-90 (ІСО/МЕК 5807-85)	Схеми алгоритмів програм, даних і систем. Умовні позначення і правила виконання

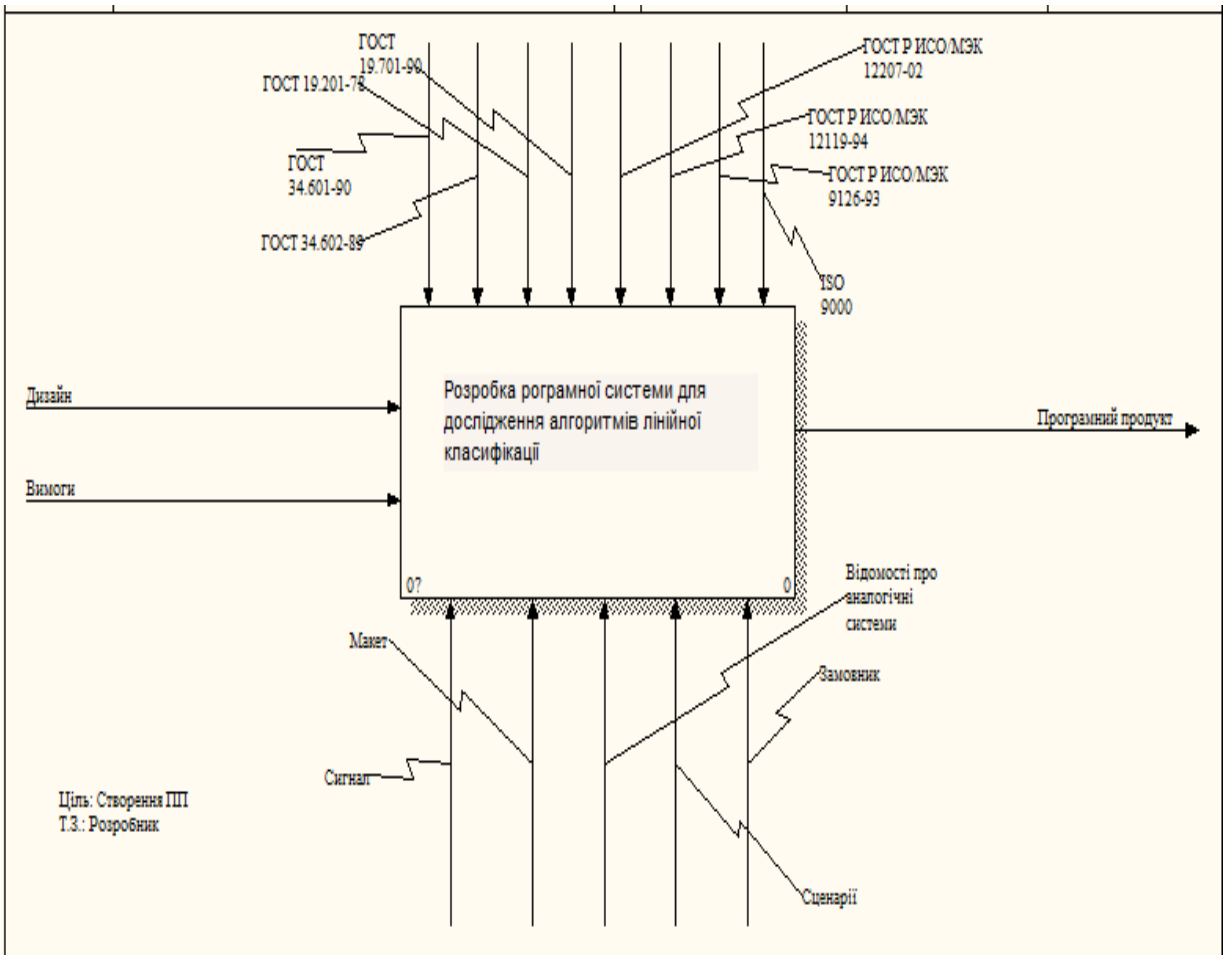


Рисунок 2.10. Контекстна діаграма.

На вхід подаються вимоги замовника до ПП та вимоги до дизайну. Ресурсами виступають макет програмного продукту, сценарії, відомості про аналогічні системи, замовник. Виходом є власне ПП.

2.4.5. Діаграма декомпозиції IDEF0

Діаграма першого рівня декомпозиції A0, а також всі наступні діаграми декомпозиції, надають інтерфейсі обмеження (контекст) для дочірніх діаграм [43].

Було вирішено розбити створення програмного продукту на наступні складові: аналіз, проектування, реалізація та тестування, що зображено на рис. 2.11.

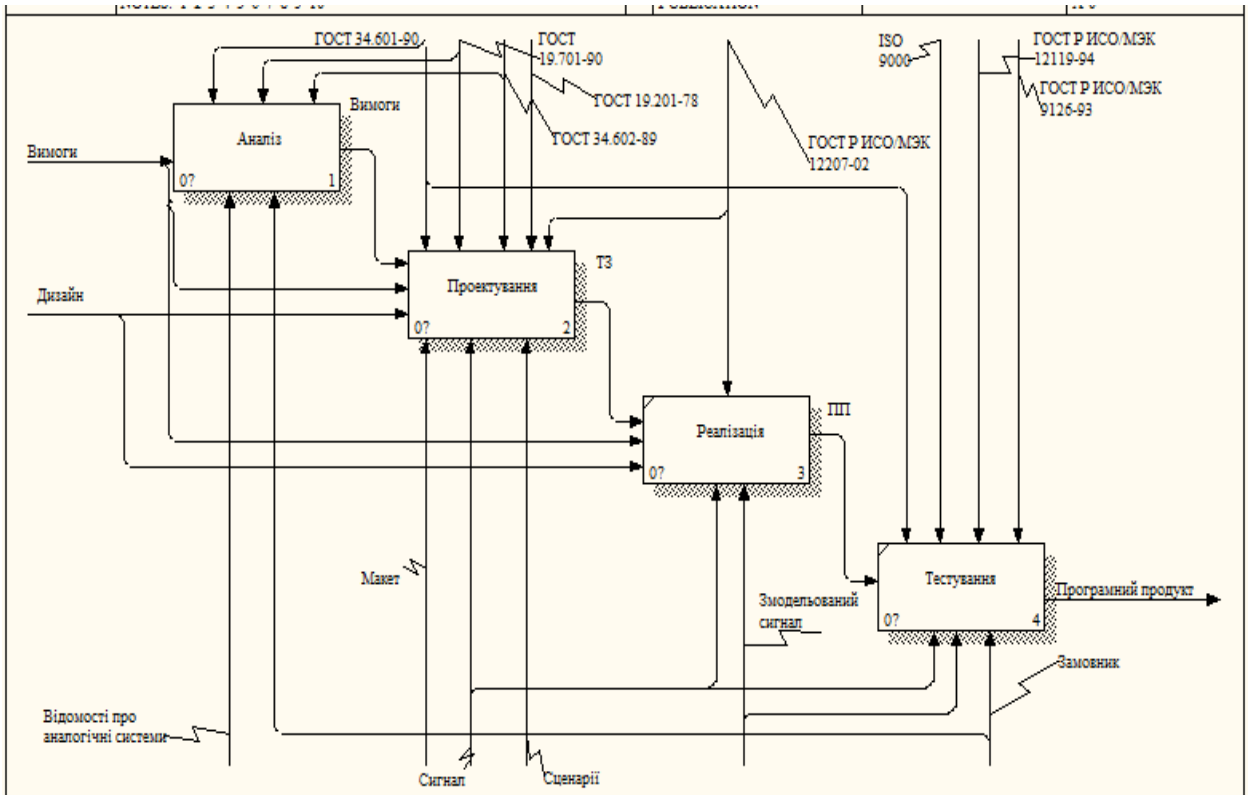


Рисунок 2.11. Діаграма декомпозиції IDEF0.

На рис. 2.12 зображена декомпозиція пункту Аналіз, що дозволяє більш детально розглянути його структуру.

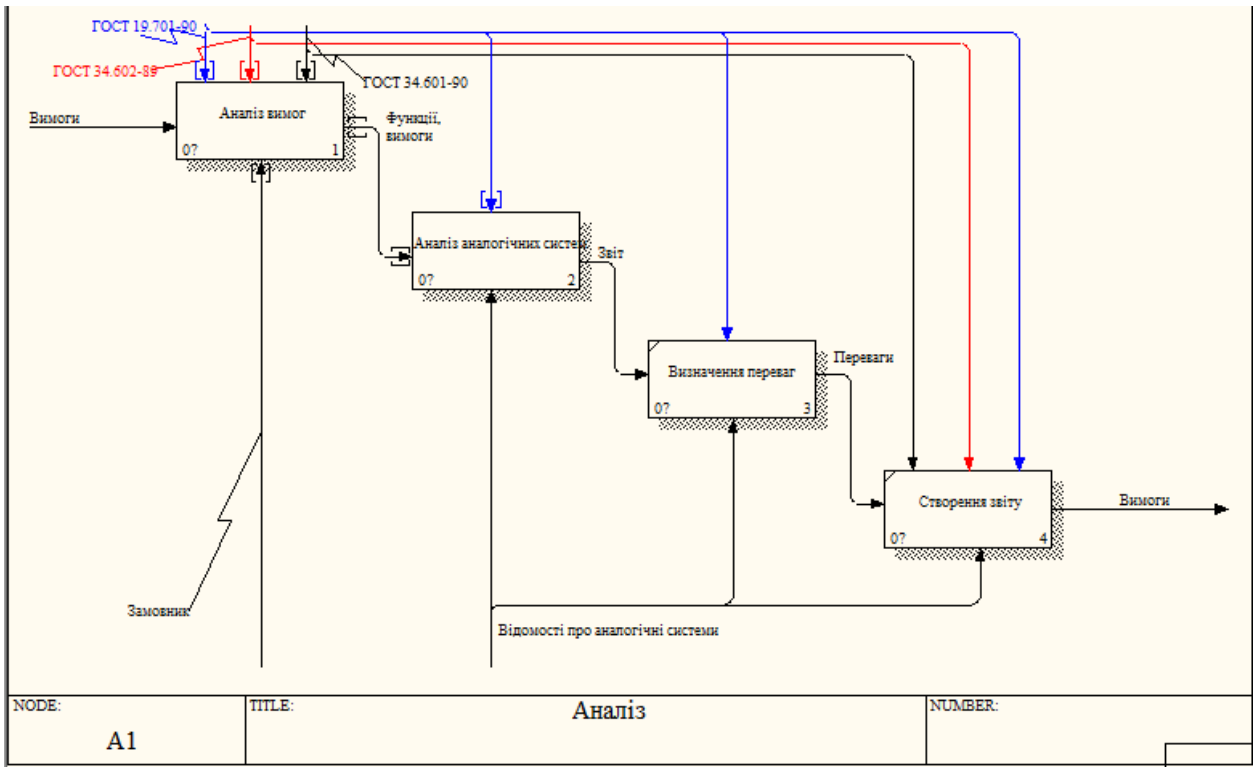


Рисунок 2.12. Діаграма декомпозиції IDEF0 (аналіз).

Не важко помітити, що аналізу вимог та аналізу використаних джерел відводиться важливе місце. Після них потрібно визначити переваги системи в порівнянні з вже існуючими.

2.4.6. Список всіх робіт

Список всіх робіт ПП наведено в таблиці. 2.3. Розглянуто програмну систему з погляду користувача та розробника. Визначено важливість етапів роботи. Детально описано, що має бути відтворено під час кожної з робіт [43].

Таблиця 2.3

Список всіх робіт

Ім’я	Важливість	«Story points»	Опис	Примітка
Графік	2	10	Як користувач я хочу бачити графічне зображення завантажених даних, щоб оцінити коректність роботи програми	Створення графічних полів
Інтерфейс	1	5	Як користувач я хочу відкрити програму, мати можливість генерувати автоматично та створювати вручну вибірки, зберігати їх	Створення зручного інтерфейсу для внесення даних та збереження

Продовж. табл. 2.3

Ім’я	Важливість	«Story points»	Опис	Примітка
Програмна частина	5	3	Як розробник я хочу реалізувати зручну програму для дослідження збіжності алгоритмів лінійної класифікації	Створення програмної системи

2.4.7. Методологія IDEF3

Для опису логіки взаємодії інформаційних потоків більше підходить IDEF3, так звана workflow diagramming, що являє собою методологію моделювання, яка використовує графічний опис інформаційних потоків, стосунків між процесами обробки інформації та об'єктів, що є частиною цих процесів (рис. 2.13) [44].

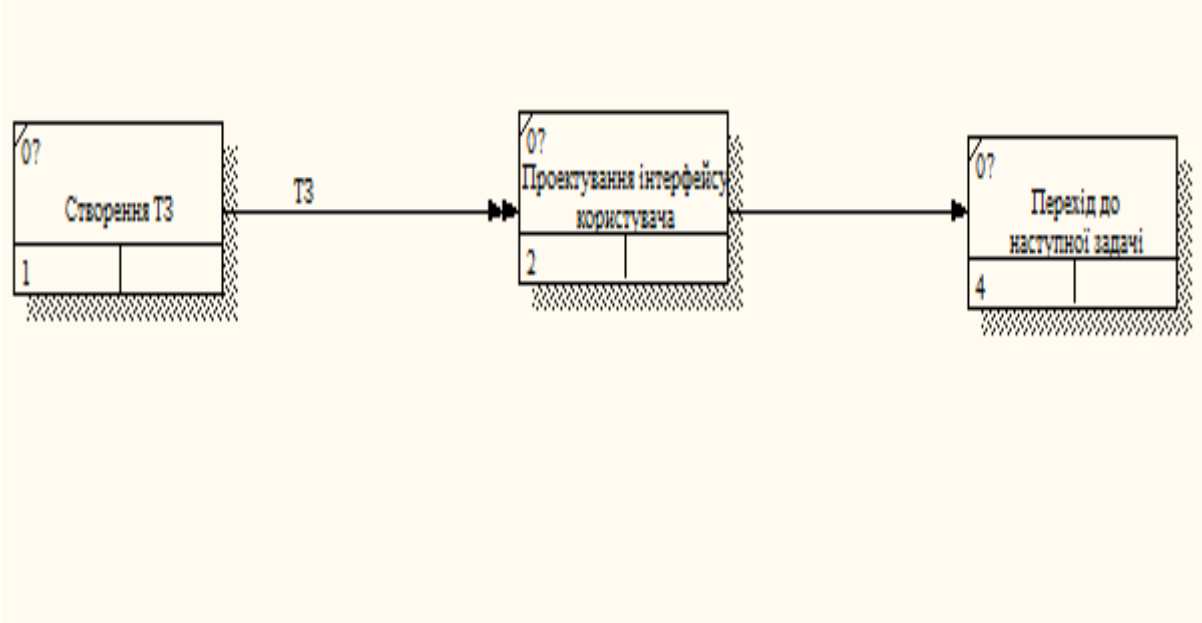


Рисунок 2.13. Діаграма IDEF3 (проектування інтерфейсу).

2.4.8. Методологія DFD

Найважливішим способом опису процесу є діаграми потоків даних (інформації) DFD (Data Flow Diagram). На рис.2.14. зображено декомпозицію пункту «Аналіз вимог». Зовнішнім посиланням тут виступає замовник, котрий і надає свої вимоги до програмного продукту. Сховищем даних є репозиторій ГОСТів [43].

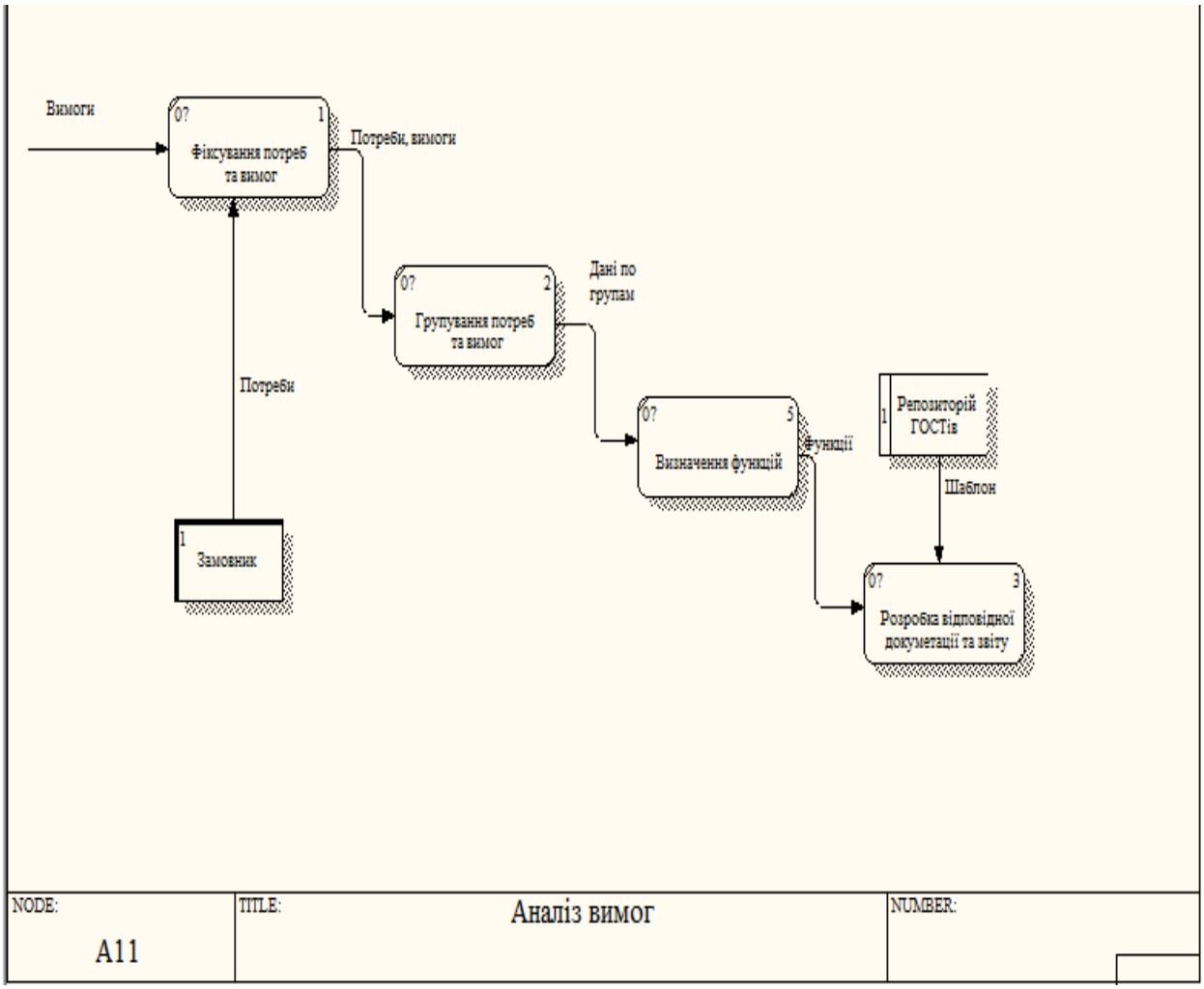


Рисунок 2.14. Діаграма DFD (аналіз вимог).

На рис. 2.15 зображено декомпозицію пункту «Аналіз аналогічних систем».

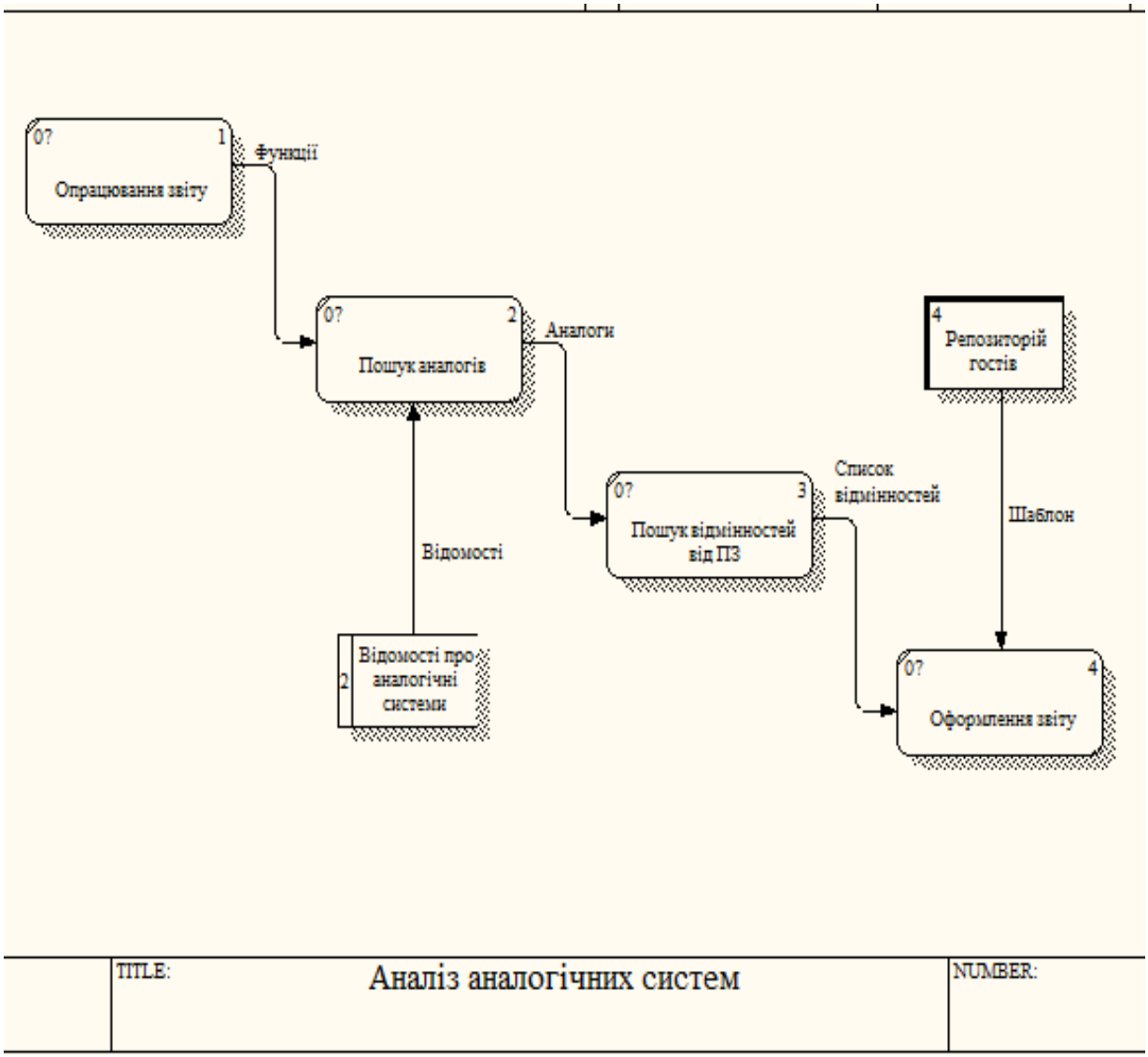


Рисунок 2.15. Діаграма DFD (аналіз аналогічних систем).

На рис. 2.16 зображена діаграма декомпозицій «Проектування», яка має такі основні блоки:

- проектування інтерфейсу;
- проектування архітектури.

Для проектування інтерфейсу необхідно мати затверджений дизайн замовником та створений на основі побажань замовника макет.

Під час проектування архітектури необхідно знати вимоги замовника та можливі сценарії роботи з програмною системою.

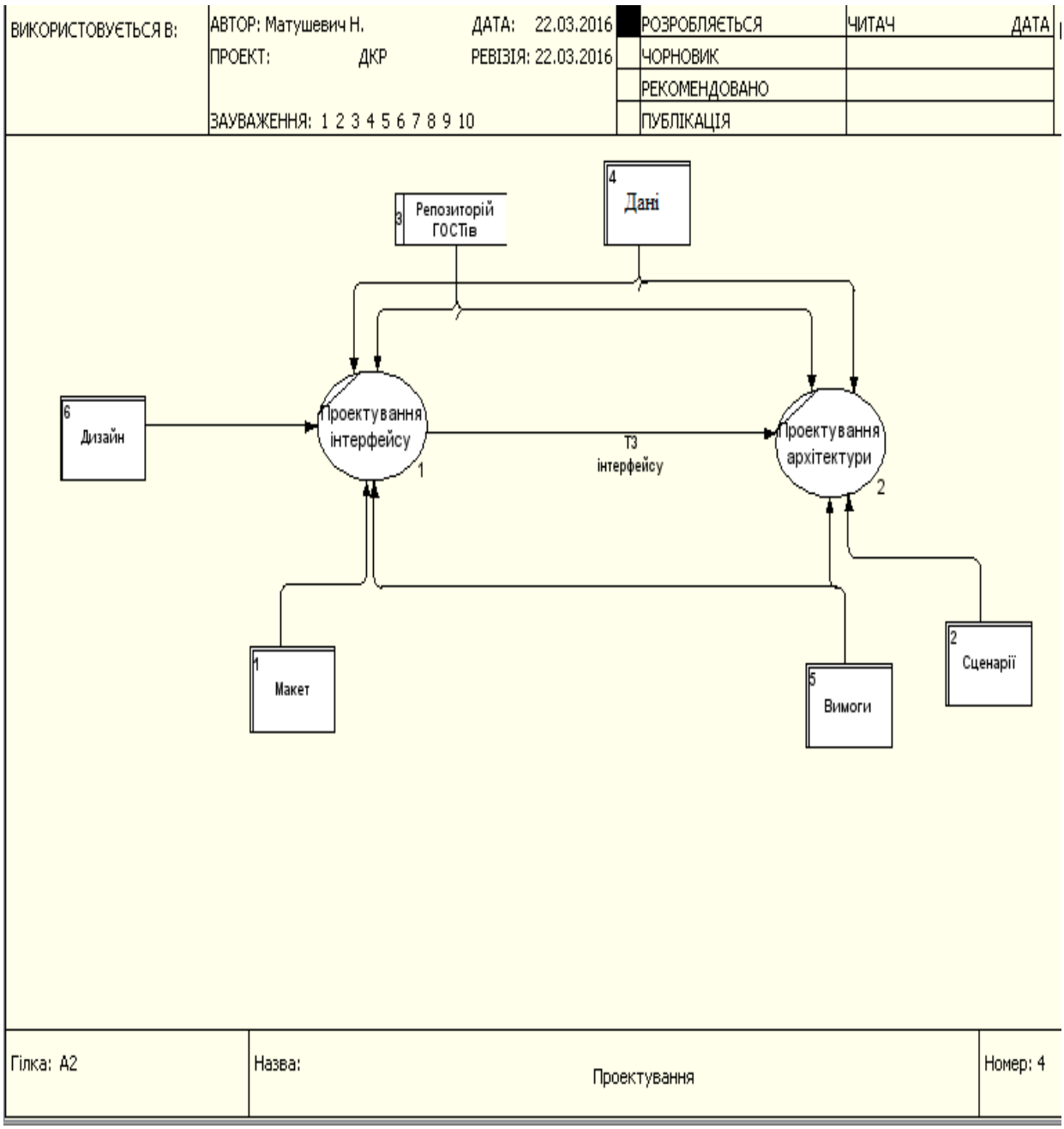


Рисунок 2.16. Діаграма декомпозиції «Проектування» (DFD).

Далі черга за реалізацією (рис. 2.17), яка складається з наступних блоків:

- реалізація інтерфейсу, модель котрого була побудована на етапі проектування;
- реалізація алгоритмів Розенблата та Козинця;

- реалізація виведення всіх зображень, до яких належать поля з вибірками та побудованими прямими для кожного з алгоритмів та стовпчикові діаграми з результатами.

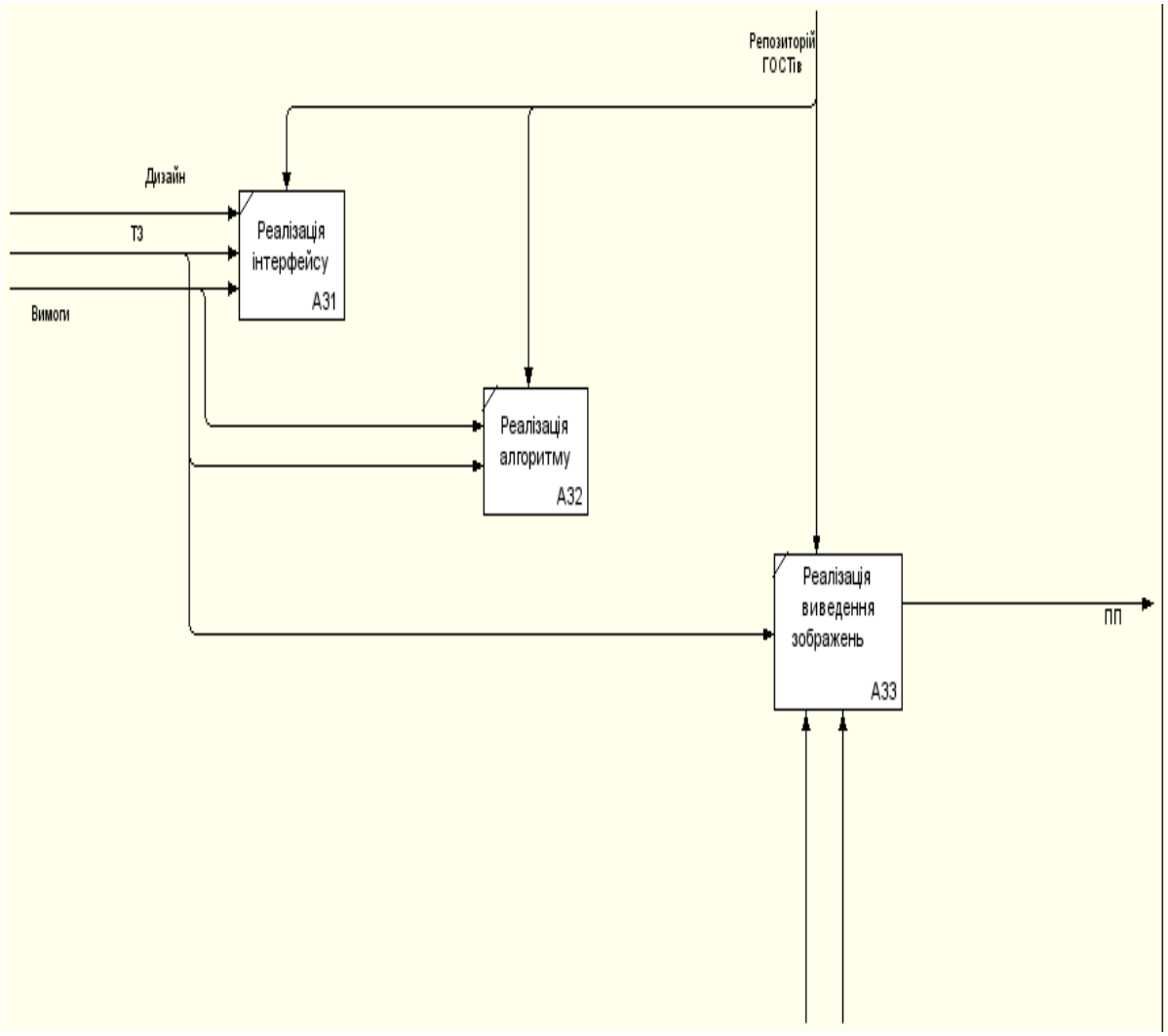


Рисунок 2.17. Діаграма декомпозиції «Реалізація» (IDEF0).

Чи не найважливіше місце в розробці програмної системи варто віддавати тестуванню (рис. 2.18). Оскільки необхідно створити якісну інструкцію з використання, протестувати інтерфейс та власне роботу системи. Крім того варто зробити гарний обробник помилок, особливо під час зчитування/запису файлу.

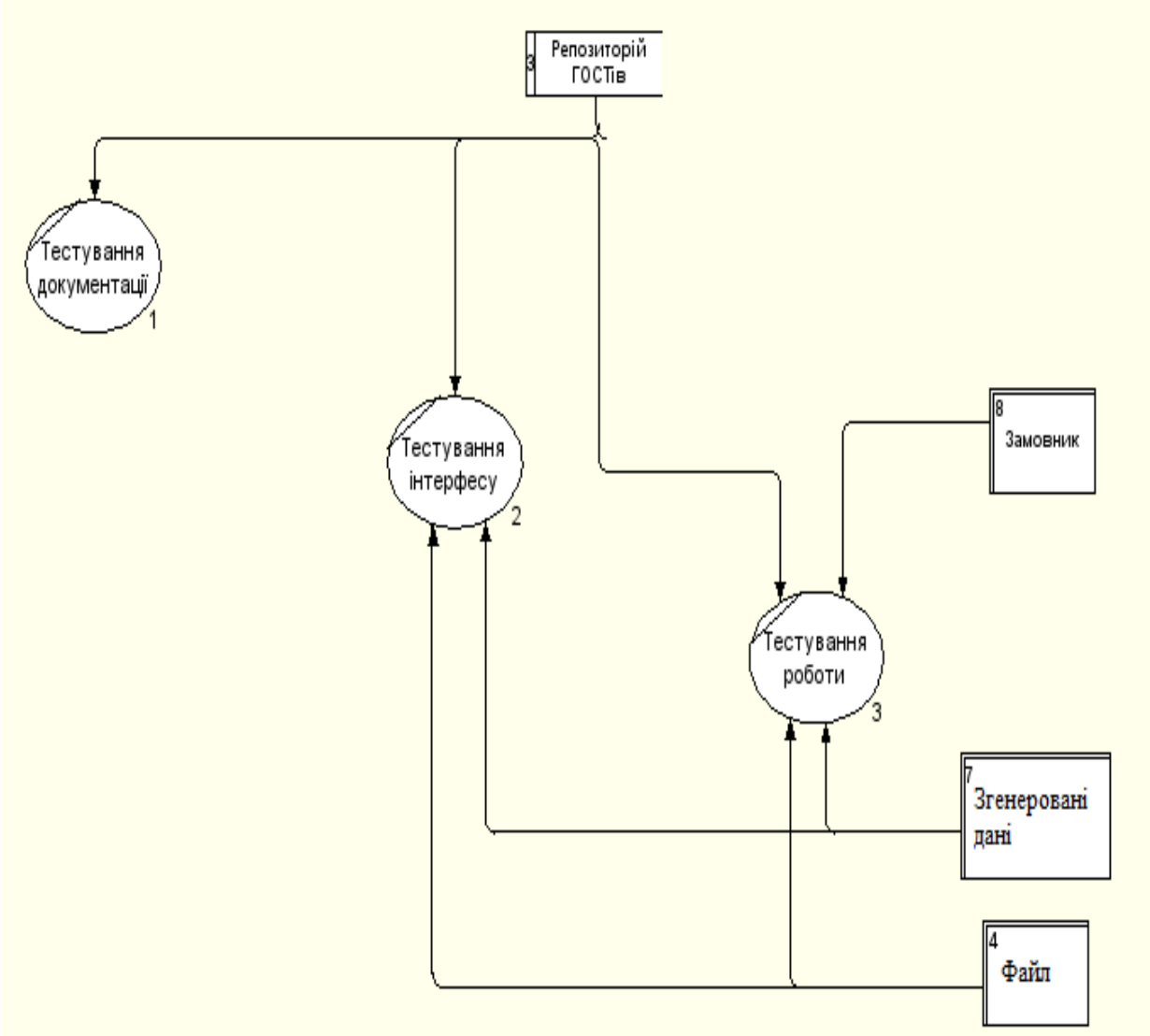


Рисунок 2.18. Діаграма декомпозиції «Тестування» (DFD).

2.4.9. Діаграма класів

Діаграми класів (Class diagrams) – головний тип діаграм UML, які відображають логічну структуру програмної системи та суттєво впливають на процес генерації програмного коду. Основними елементами діаграми класів у Rational Rose є безпосередньо класи (classes) та відношення між ними (relations) [44].

Було виділено наступні класи: ПП, Алгоритми, Користувач, Вхідний сигнал, Вихідний сигнал. Їхні методи, атрибути та відношення між ними наведені на рис. 2.19.

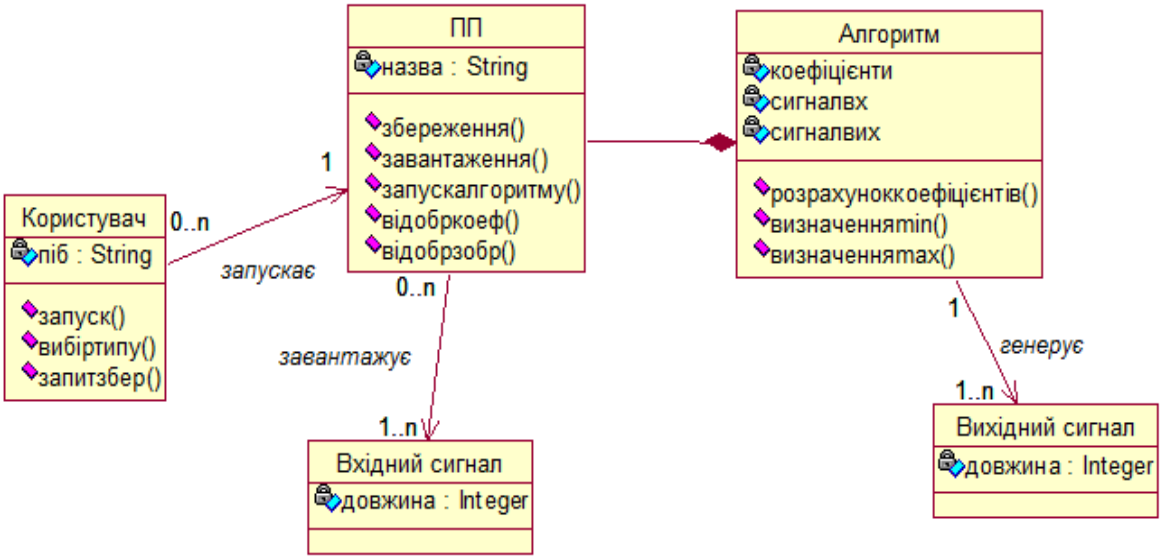


Рисунок 2.19. Діаграма класів.

2.4.10. Діаграми реалізації

Діаграма компонент – в UML, діаграма, на якій відображаються компоненти, залежності та зв'язки між ними [45].

Діаграма компонент (рис. 2.20) відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись. Модуль програмного забезпечення може бути представлено як компоненту. Деякі компоненти існують під час компіляції, деякі – під час компонування, а деякі під час роботи програми [45].

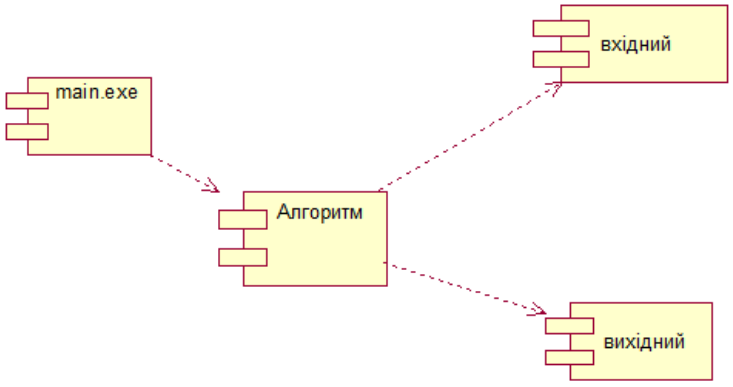


Рисунок 2.20. Діаграма компонентів.

2.4.11. Діаграма діяльності

Діаграми діяльності (activity diagrams) відображають послідовність дій, що виконується в процесі реалізації певного варіанта використання або функціонування системи в цілому (рис. 2.21) [46].

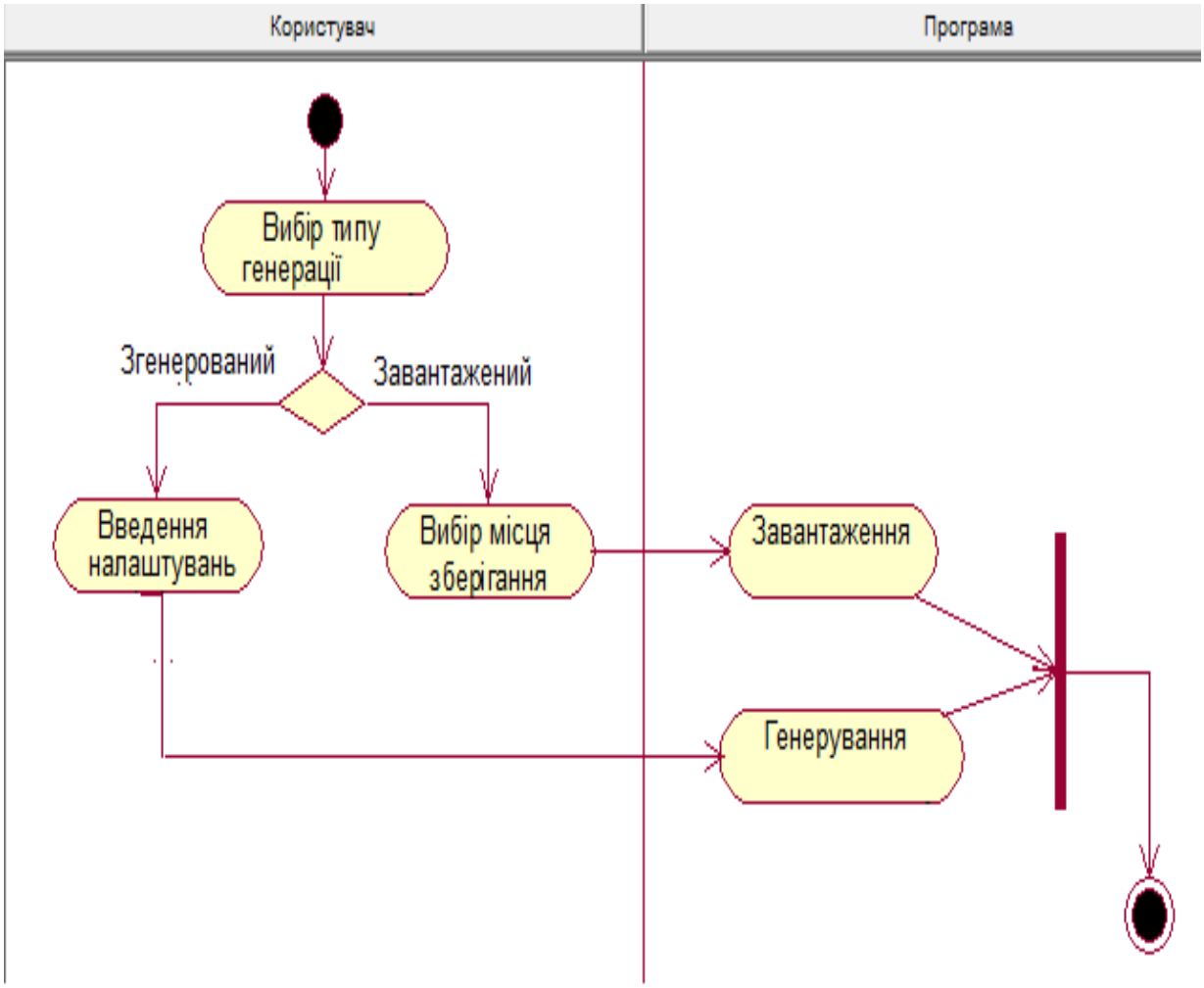


Рисунок 2.21. Діаграма діяльності «Вибір типу генерації».

Після запуску програми користувач обирає використовувати завантажену з файлу вибірку чи згенерувати її автоматично чи мануально. У разі використання генерованого користувач має налаштувати програму для кращої вибірки, після чого сигнал генерується програмою. Якщо ж вибрали завантажений, то програма завантажує файл. Після закінчення виконання операцій гілки поєднуються в один потік, в якому програма запускає алгоритми, алгоритми знаходять коефіцієнти прямих класифікації, генерує лінії. Програма виводить зображення (рис. 2.22).

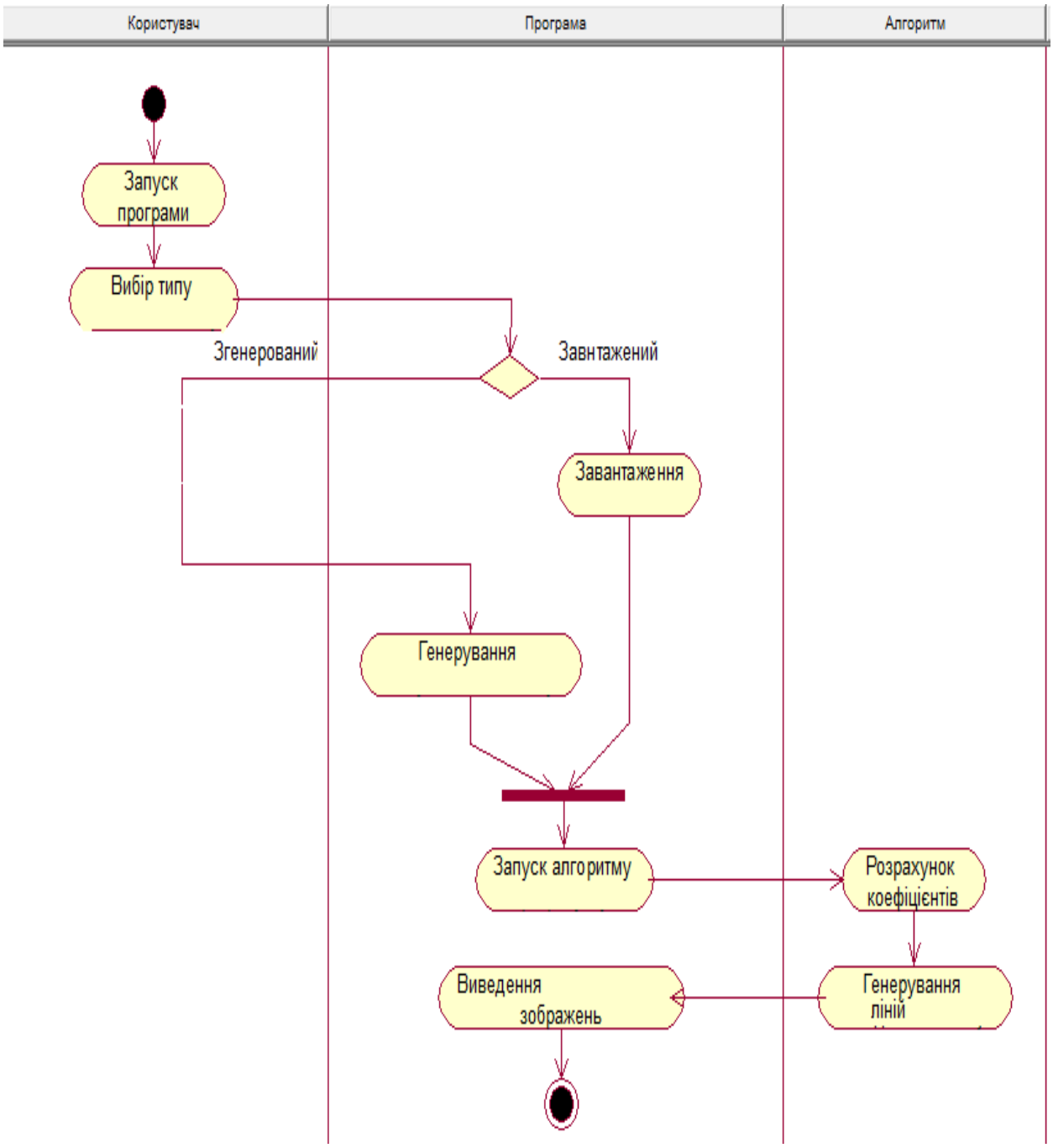


Рисунок 2.22. Діаграма діяльності ПП.

2.4.12. Діаграма дерева вузлів

Діаграма дерева вузлів показує ієрархічну залежність робіт, але не взаємозв'язки між роботами (рис. 2.23) [45].

Рисунок 2.23. Діаграма дерева вузлів.

2.4.13. Діаграма варіантів

Діаграми варіантів використання (usecase diagrams) використовуються для відображення сценаріїв використання системи (usecases) та користувачів системи (actors), які використовують її функції [46].

Актором виступає користувач. Сценарії наступні: генерація даних трьома різними способами:

• автоматично;

• вручну шляхом вказівки точок вибірки, що відповідають різним класам, на площині;

• завантаживши з файлу.

Під час одиничного запуску програми генеруються безліч точок двох класів, які точно можуть бути розділені прямою лінією, і паралельно запускаються алгоритми навчання Розенблатта і Козинця (рис. 2.24).

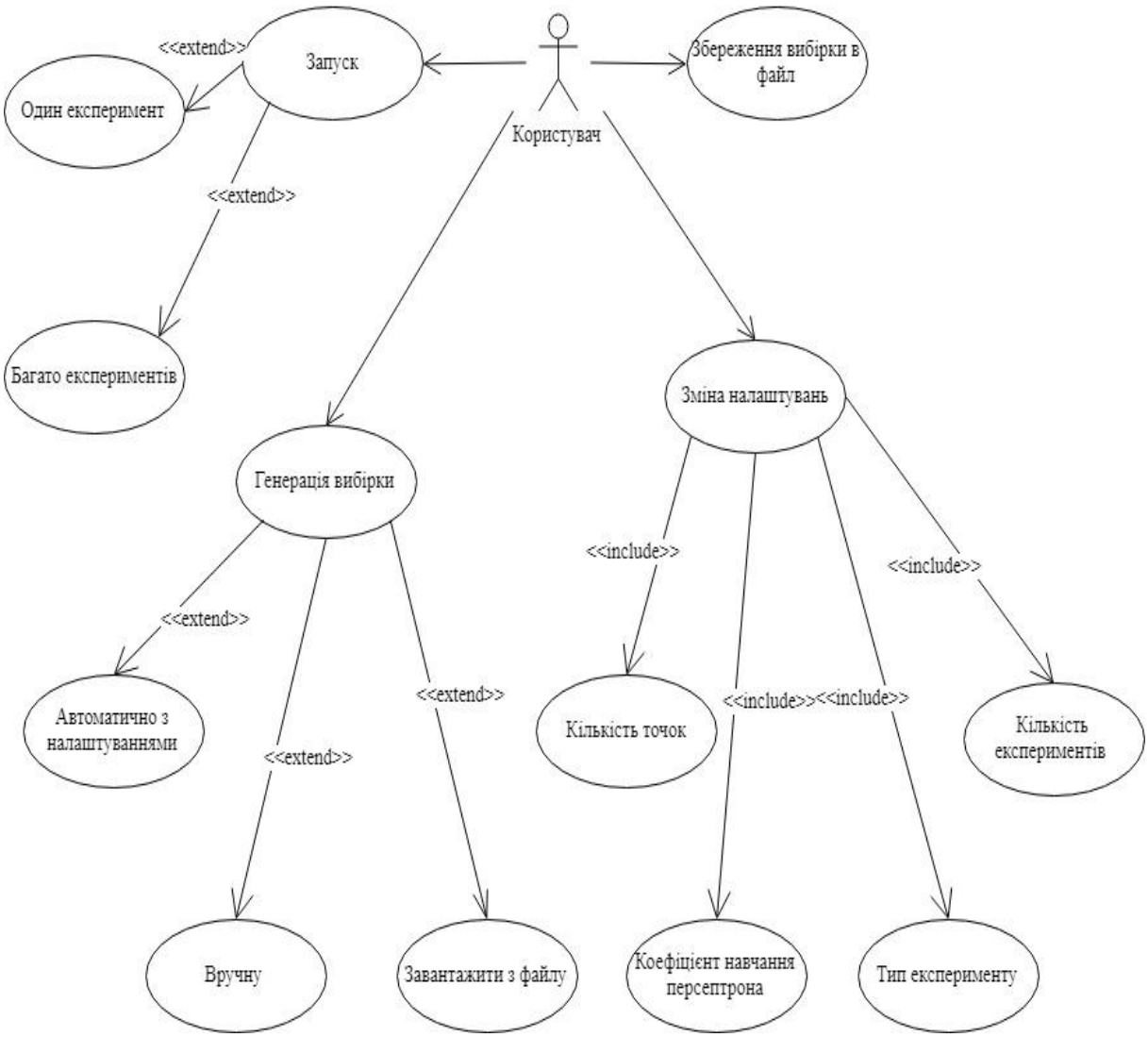


Рисунок 2.24. Use-case діаграма.

В кінці експерименту відображаються розділяючі прямі і гістограма кількості ітерацій, витрачених кожним з алгоритмів.

При множинних експериментах є ряд додаткових функцій, зокрема відображення гістограми процентного співвідношення кількості итерацій витрачених кожним алгоритмом при навчанні. Це дає можливість оцінювати і порівнювати ймовірності успішного завершення навчання і тим самим визначити лідера при конкретних настройках системи (рис. 2.25) [48].

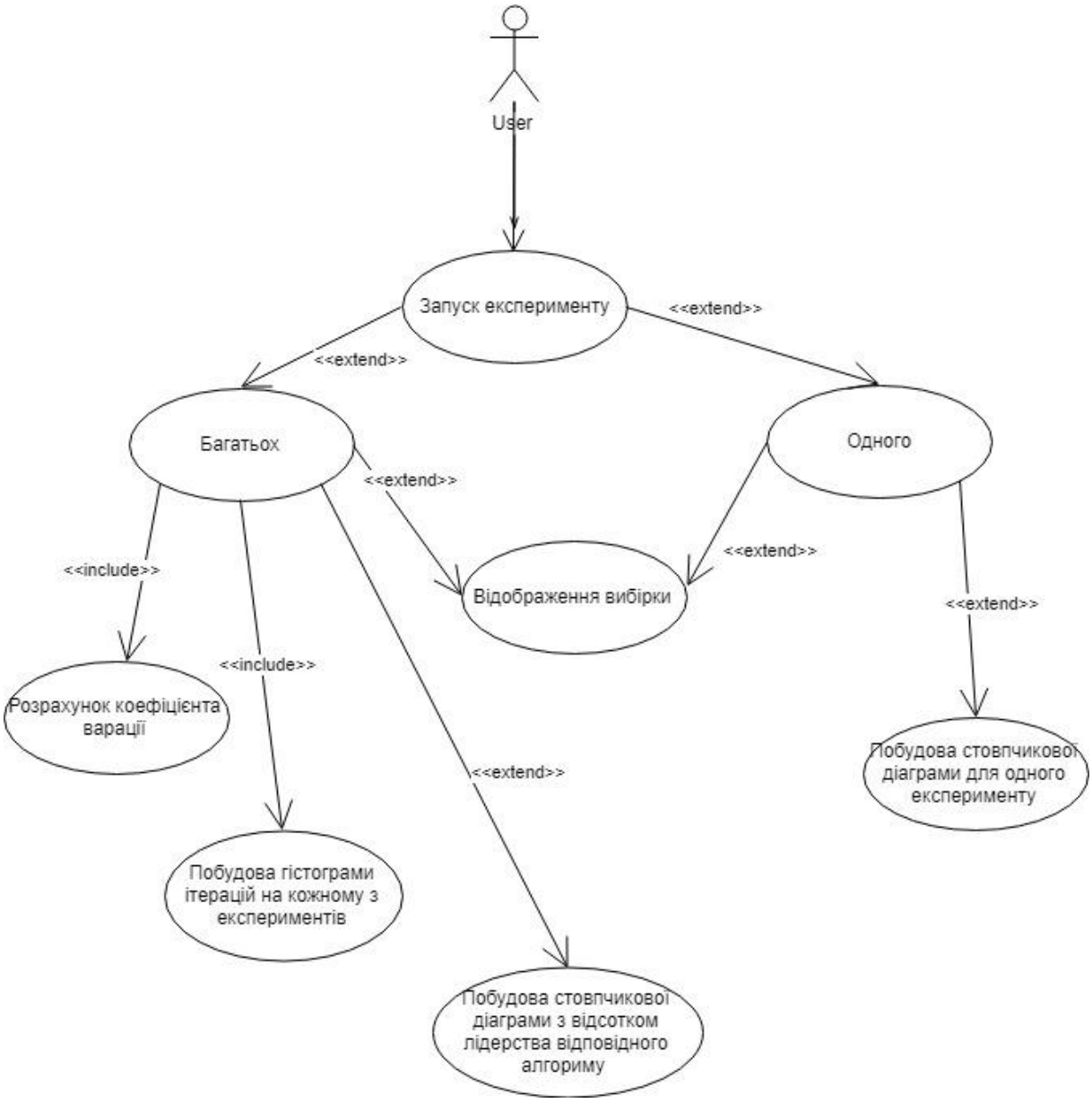


Рисунок 2.25. Use-case діаграма.

2.2.14. Діаграми послідовності

Також наявна можливість проводити розрахунок коефіцієнта варіації кількості итерацій, витрачених алгоритмами при множинних експериментах.

Для ілюстрації на рис. 2.26 представлена діаграма послідовностей (sequence diagram), яка пояснює деталі зв'язків між основними блоками програми:

- інтерфейсом;
- алгоритмом генерації;

- блоком відображення;
- алгоритмом Розенблата;
- алгоритмом Козинця.

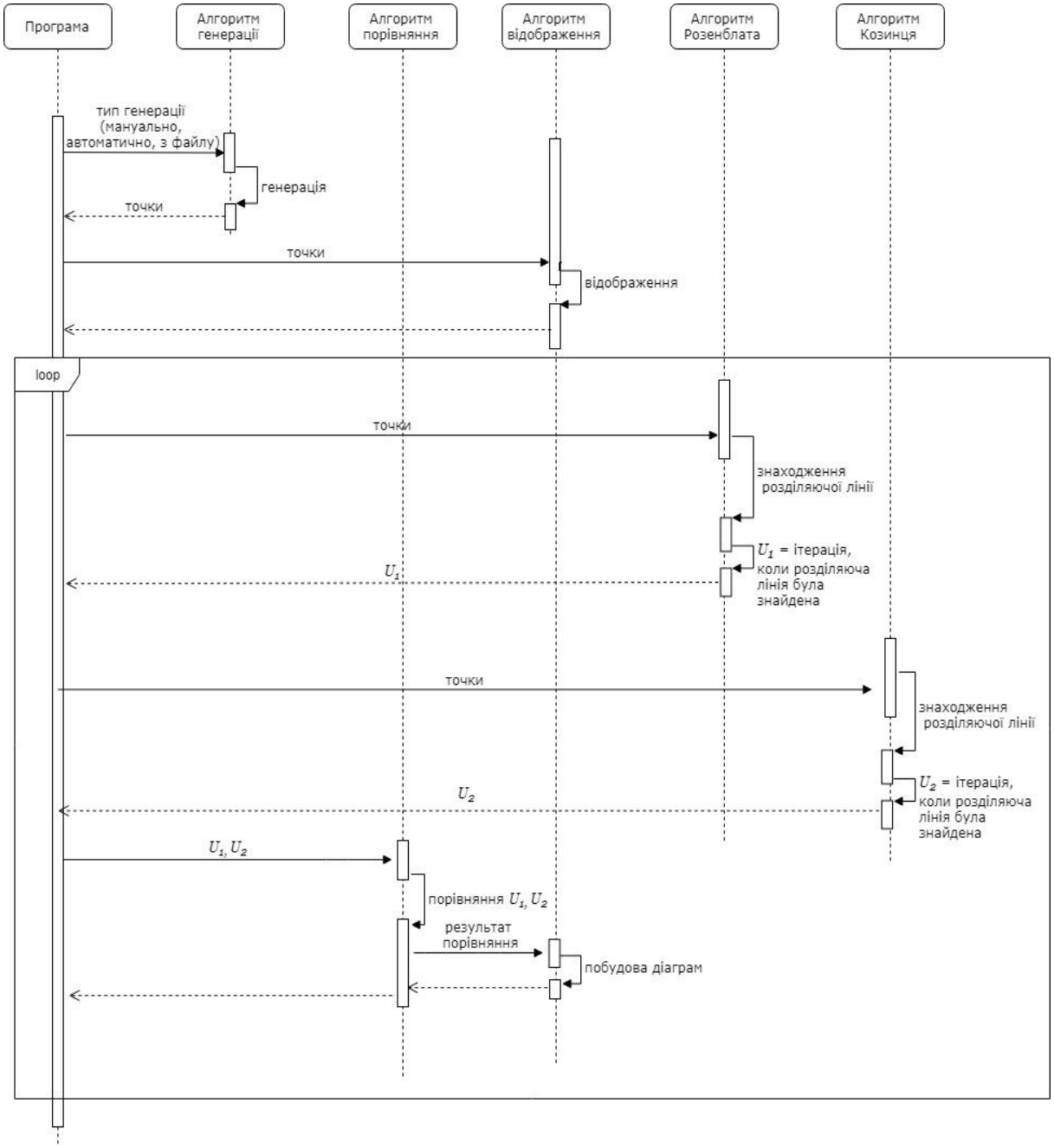


Рисунок 2.26. Діаграма послідовності ПП.

При кожному запуску процедур, що реалізують алгоритми Розенблата і Козинця обчислюються числа ітерацій і до зупинки роботи алгоритмом Розенблата і Козинця відповідно. Моменти зупинки визначає умова

$$\left\langle w, x_j \right\rangle \cdot c_j > 0, \quad j = 1,...n, \tag{2.14}$$

яка відповідно до (2.2), (2.4), (2.5) свідчить про те, що при поточному значенні вектора параметрів всі точки навчальної вибірки, відповідні класам V_1 і V_2, повністю розділені лінійною дискримінантною функцією.

На підставі порівняння чисел U_1 і U_2 визначається лідер в поточному експерименті:

Лідер алгоритм Розенблата, якщо $U_1 < U_2$**,**

Лідер алгоритм Козинця, якщо $U_2 < U_1$**.**

Результати порівняння візуалізується відповідною стовпчиковою діаграмою.

Детальніше розглянемо генерацію точок вибірки (рис. 2.27). Користувач має можливість обирати зручний для нього тип генерації вибірки:

- завантажити з файлу, котрий відкриває діалог з файловою системою, для вибору файлу;
- мануально, котрий дозволяє вручну задати положення всіх точок вибірки на площині;
- автоматично, котрий програмно генерує випадкові положення точок спираючись на налаштування (кількість точок та ін.).

В залежності від вибору програма чи відкриває обраний файл, зчитує точки, чи генерує їх автоматично, або передає управління користувачеві для задання їх мануально [48].

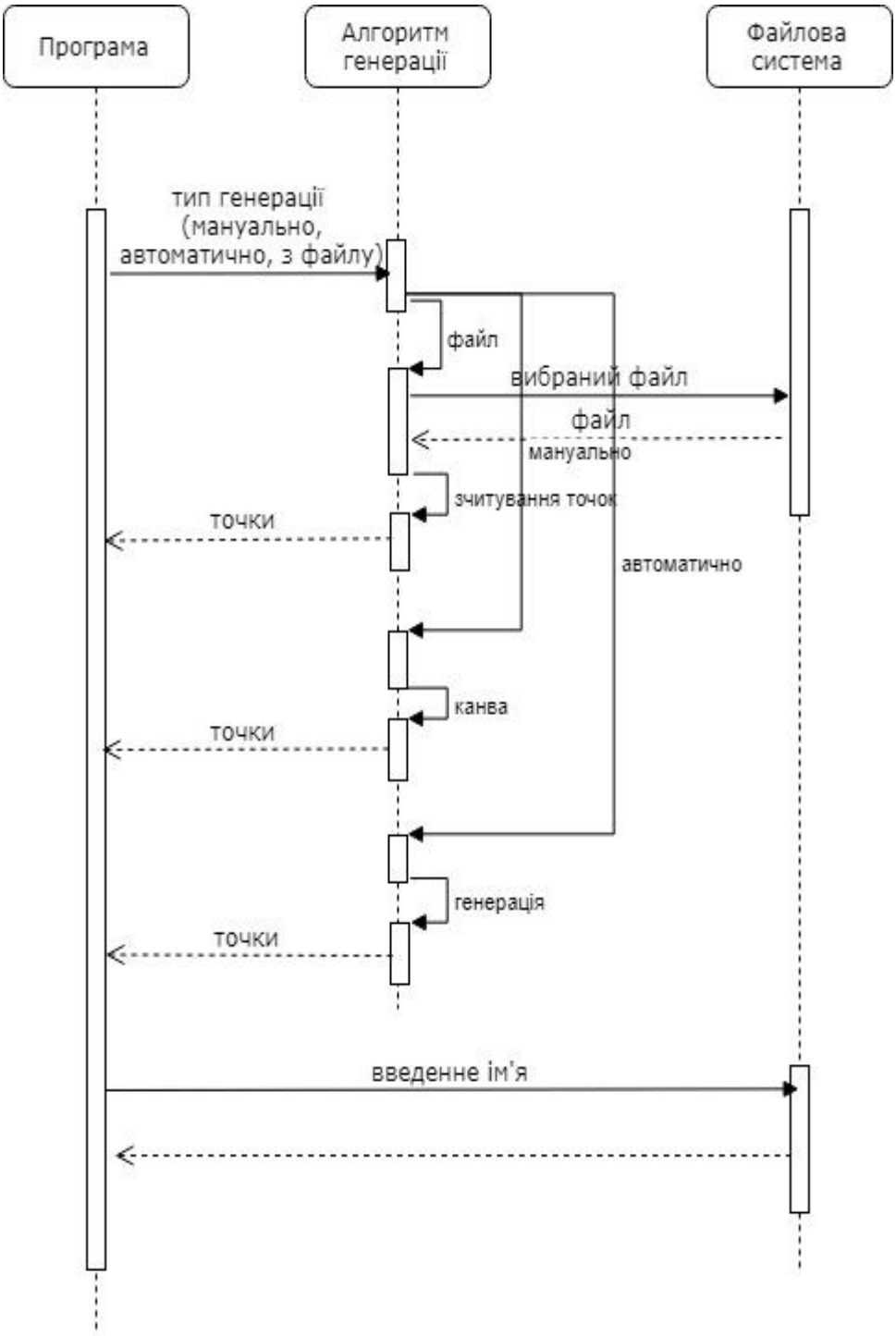


Рисунок 2.27. Діаграма послідовності блока генерації.

2.5. Реалізація програмного продукту

Розроблений програмний продукт написано мовою C# з використанням MS Visual Studio 2017 [39].

Метою розробки є виконання програмної реалізації алгоритмів Розенблата та Козинця. Програмне забезпечення дозволить досліджувати збіжність зазначених алгоритмів.

Розроблений інтерфейс користувача є досить простим і інтуїтивно зрозумілим.

Вхідні дані при роботі з розробленою програмою:

- точки;

Основні можливості системи, що повинні бути розроблені:

- можливість задання точок мануально;
- можливість завантаження точок;
- можливість генерування точок автоматично
- відображення отриманих точок;
- запуск алгоритмів для класифікації;
- можливість налаштування роботи, вибору кількості експериментів та точок для генерації, налаштування алгоритму Розенблата та налаштування інтерфейсу;
- побудова стовпчикових гістограм по одному та багатьох експериментах;
- розрахунок коефіцієнта варіантності та побудова на його основі гістограм;
- можливість збереження вибірки;
- можливість використання програмного продукту на різних операційних системах.

Для реалізації програмного забезпечення було обрано MS Visual Studio 2017, тому що дане програмне забезпечення має необхідний інструментарій для виконання поставлених задач замовником. Є зручним, та розповсюджується для студентів безкоштовно.

Висновки до розділу 2

В ході проектування ПП була створена ієрархічна структура та розрахована нев’язка, яка свідчить про добре сплановану систему ПП, яку буде доцільно розробляти. Здійснено проектування програмного продукту у вигляді діаграм, які описують розробку та функціонування програми.

РОЗДІЛ 3
РОБОТА З РОЗРОБЛЕНИМ ПРОГРАМНИМ ПРОДУКТОМ

Після запуску програми з’являється інтерфейс представлений на рис. 3.1.

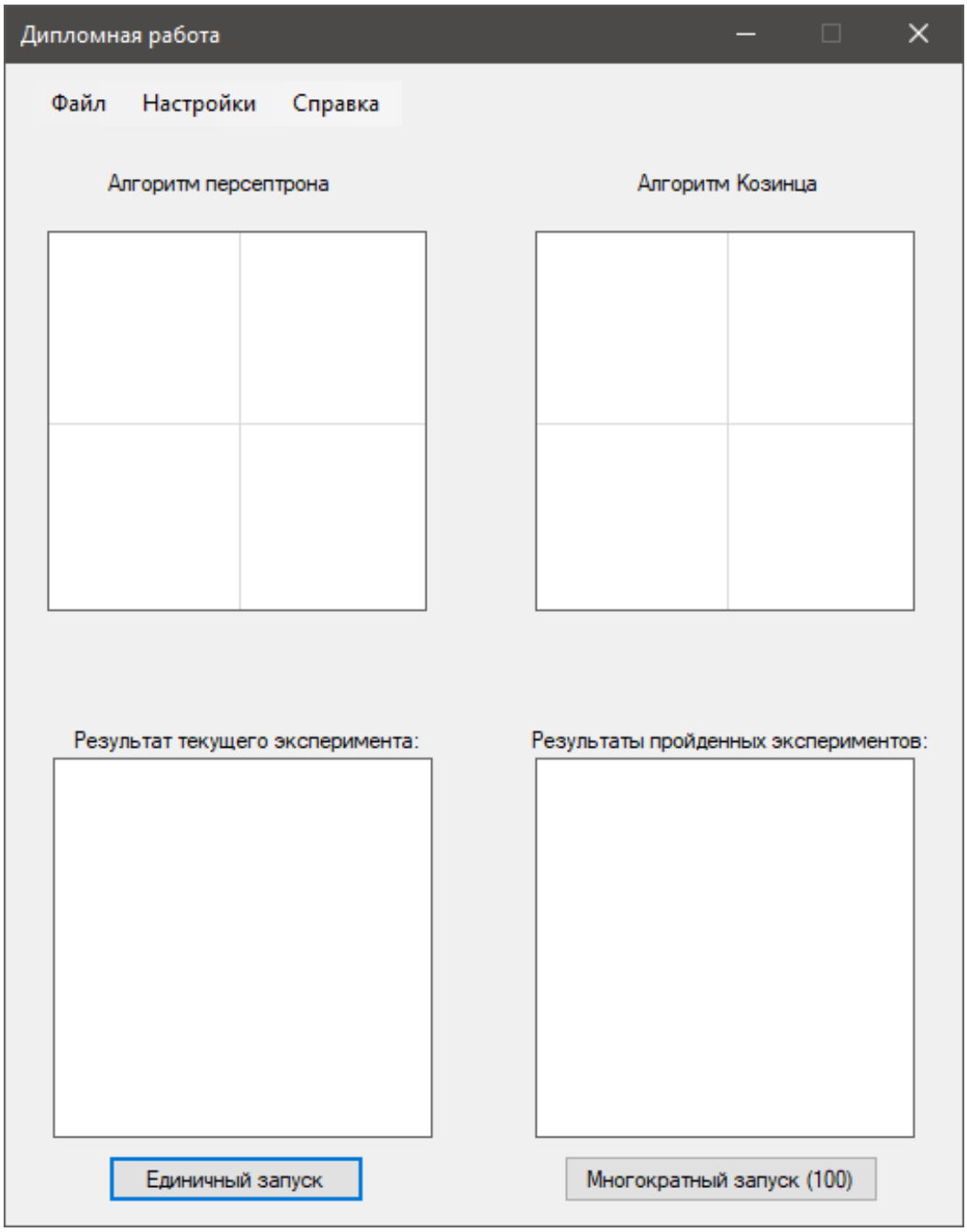


Рисунок 3.1. Інтерфейс програми.

Перш за все необхідно визначити яким чином буде отримана вибірка. В разі задання точок інтерактивно необхідно клацнути лівою (якщо

перший клас) чи правою (якщо другий) кнопкою миші на виділеній області (рис. 3.2).

Рисунок 3.2. Задання точок мануально.

У випадку автоматичної генерації точок необхідно вибрати відповідний пункт меню «Файл» (рис. 3.3).

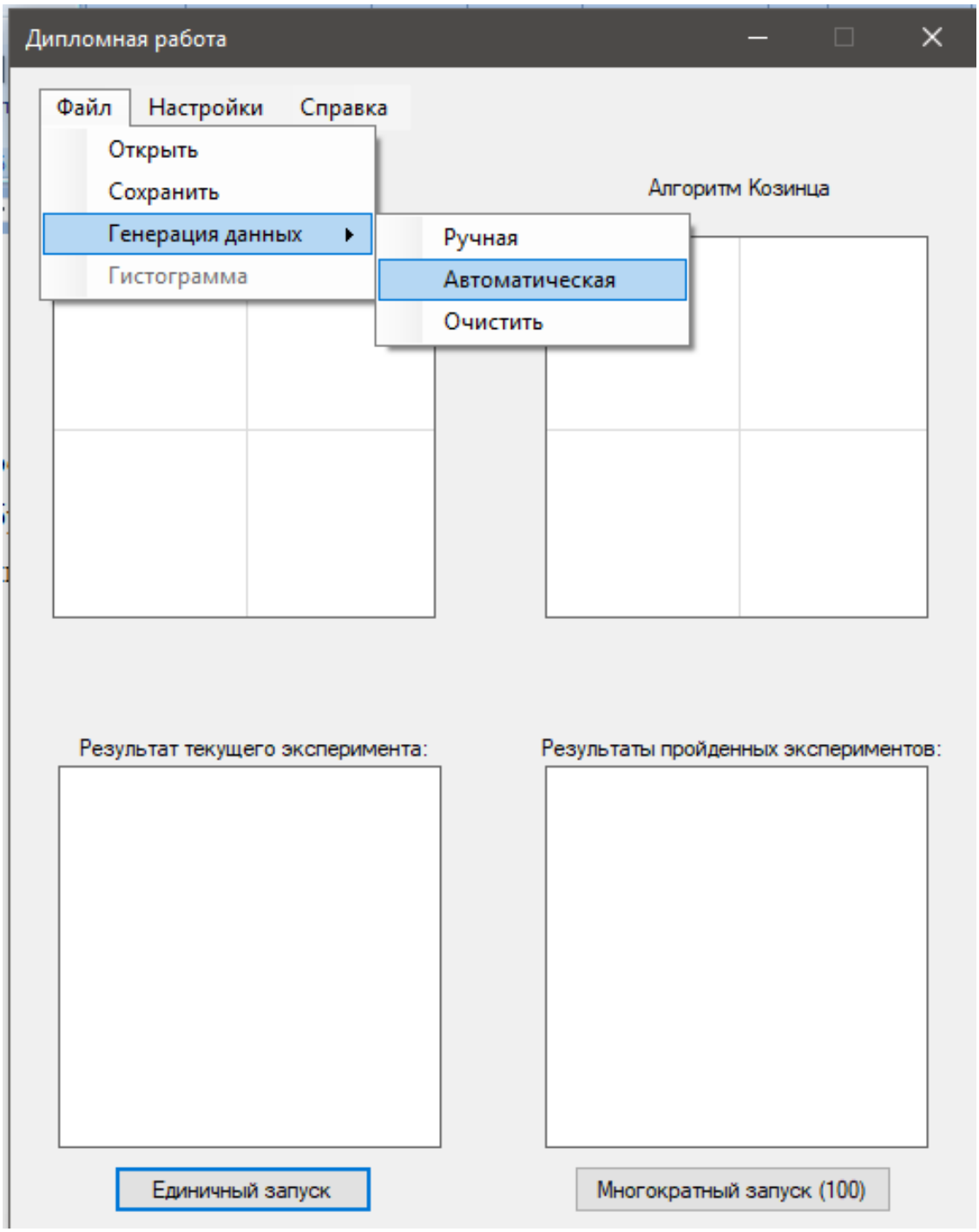


Рисунок 3.3. Автоматична генерація точок.

І останнім варіантом є можливість завантаження файлу з системи через меню «Файл»-«Открыть». Що дозволить відкрити діалог з системою та обрати файл в форматі «.txt». Після чого програма повідомить, що точки були завантажені, або причину чому не були (рис. 3.4).

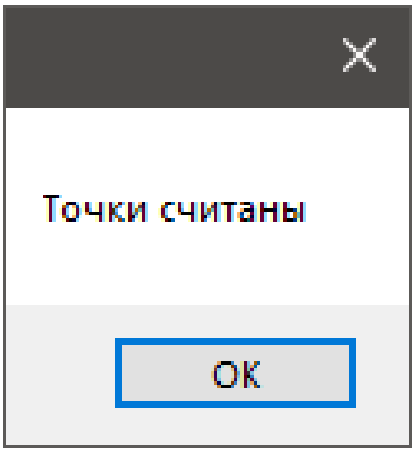


Рисунок 3.4. Сповіщення при завантаженні точок.

У вікні «Настройки» є можливість налаштування роботи програми (рис. 3.5):

- показувати / не показувати лінію за якою генерувалися автоматично точки;
- кількість точок в вибірці для автоматичного генерування (10..200) ;
- зміна коефіцієнта навчання персептрон;
- якщо багаторазовий запуск на різних даних обрано, то при кожному із послідовності запусків буде генеруватися нова послідовність, котра відмінна від попередньої;
- в зворотному випадку – вся кількість повторних запусків буде проведена на одній вибірці;
- задання кількості експериментів для багаторазового запуску;
- вповільнення відображення руху прямої під час ітерацій кожного з алгоритмів;
- відображати/не відображати перші точки, які було обрано для алгоритма Козинця.

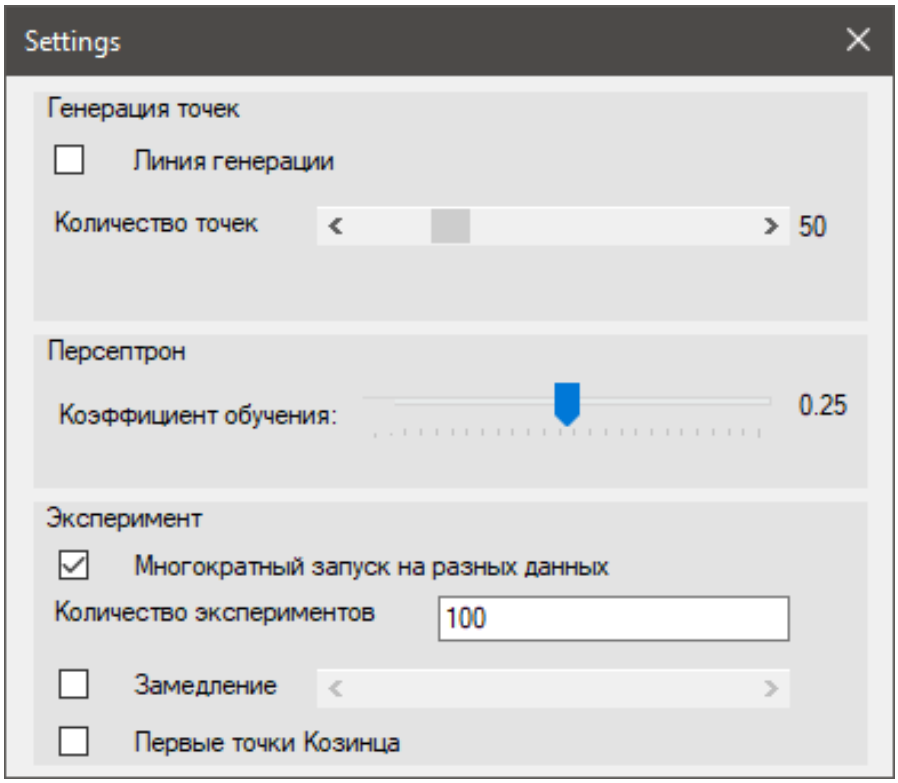


Рисунок 3.5. Вікно налаштувань.

При натисканні на «Справка» відкривається вікно з інформацією про авторів (рис. 3.6).

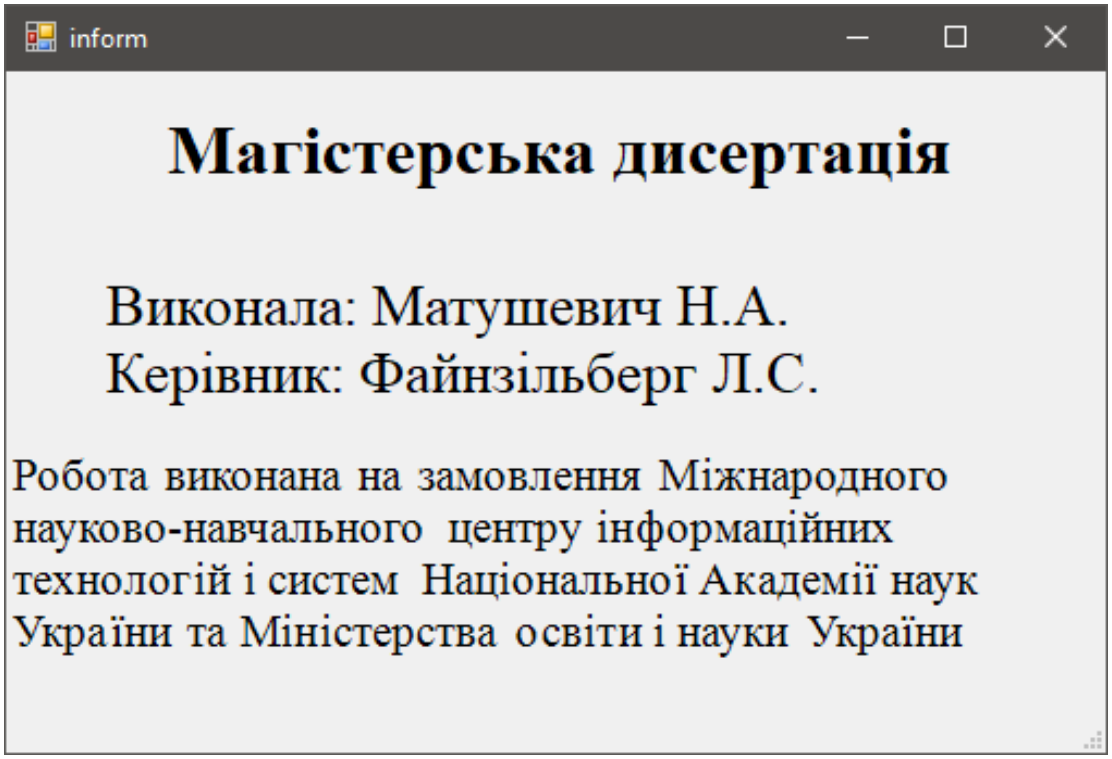


Рисунок 3.6. Вікно довідки.

Після генерації вибірки та всіх налаштувань можливо запустити алгоритми двома способами (рис. 3.7). Перший з них дозволяє запустити один раз обидва алгоритми на одній вибірці. Інший – багаторазово відповідно до налаштувань (за промовчуванням на різних даних).

Після натискання виділеної кнопки та роботи алгоритмів відобразиться наступне:

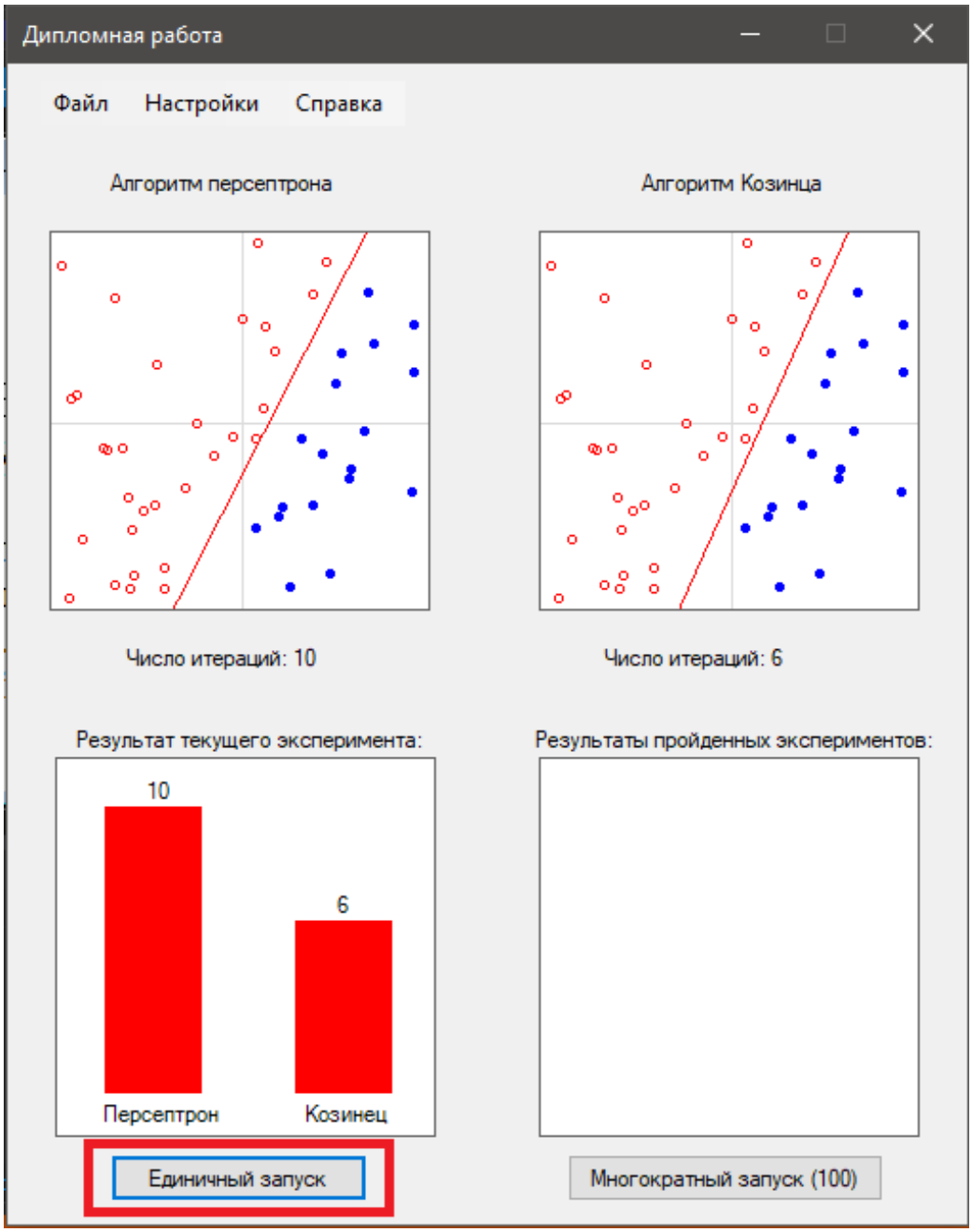


Рисунок 3.7. Перший варіант запуску.

У верхніх лівому та правому квадратах відображається вибірка та прорахована відповідно кожним з алгоритмів розділяюча пряма. В нижньому лівому квадраті відображається кількість ітерацій, котрі були необхідні кожному з алгоритмів для пошуку правильного рішення. Число над стовпчиком і вказує цю кількість.

Якщо ж натиснути «Многократный запуск» (рис. 3.8), то алгоритми будуть запущені стільки разів скільки вказано в дужках на кнопці і задано в налаштуваннях:

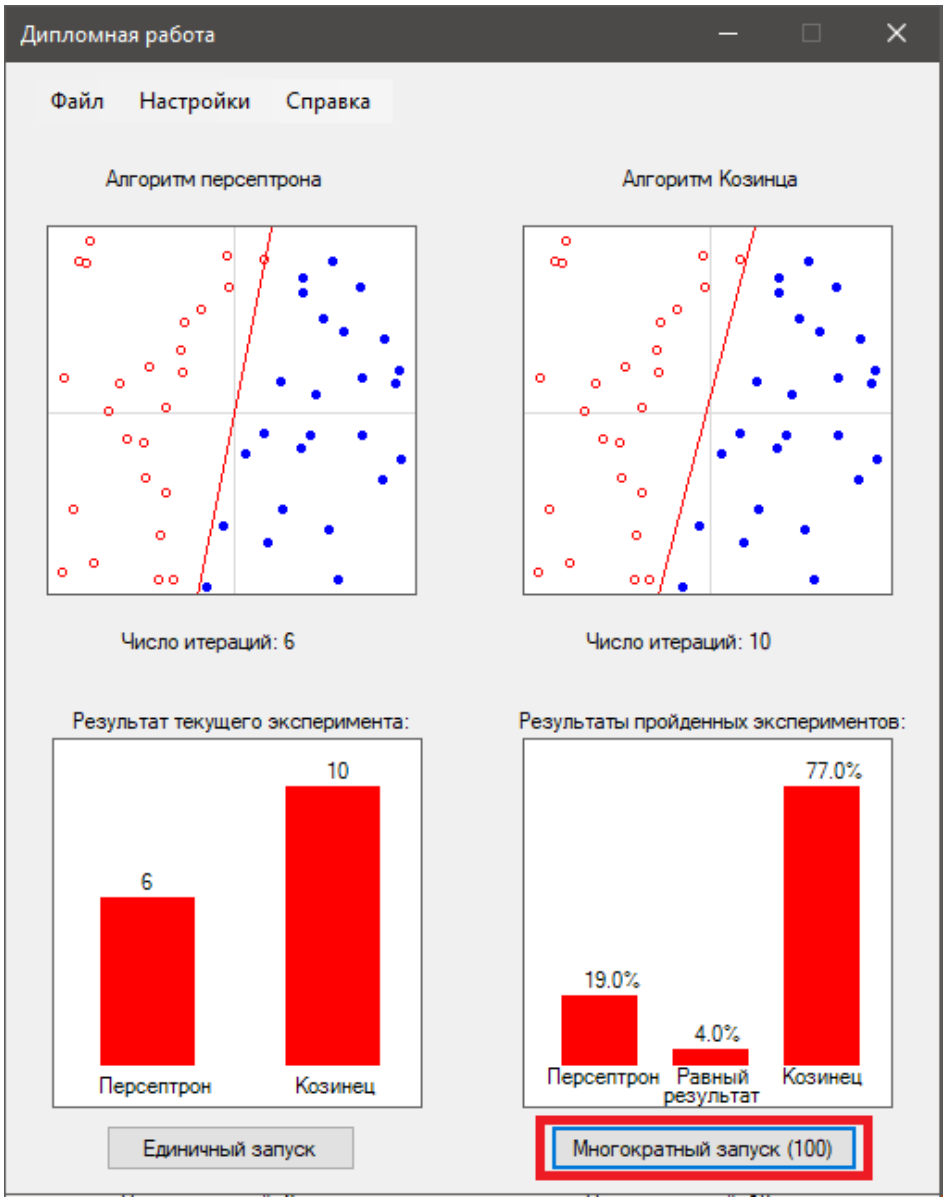


Рисунок 3.8. Другий варіант запуску.

При багаторазовому запуску результат кожного з запусків відображається в верхніх квадратах. Після закінчення циклу залишається останній результат. Також в режимі реального часу змінюють свою висоту стовпчикові діаграми правого нижнього квадрату. В них відображається процент випадків коли той, чи інший алгоритм знайшов вирішення за меншу кількість ітерацій. Також наявний стовпчик для випадків, коли обом алгоритмам була необхідна однакова кількість ітерацій.

Після виконання багаторазового запуску стає активним ще один пункт з меню «Файл», а саме гістограма (рис. 3.9).

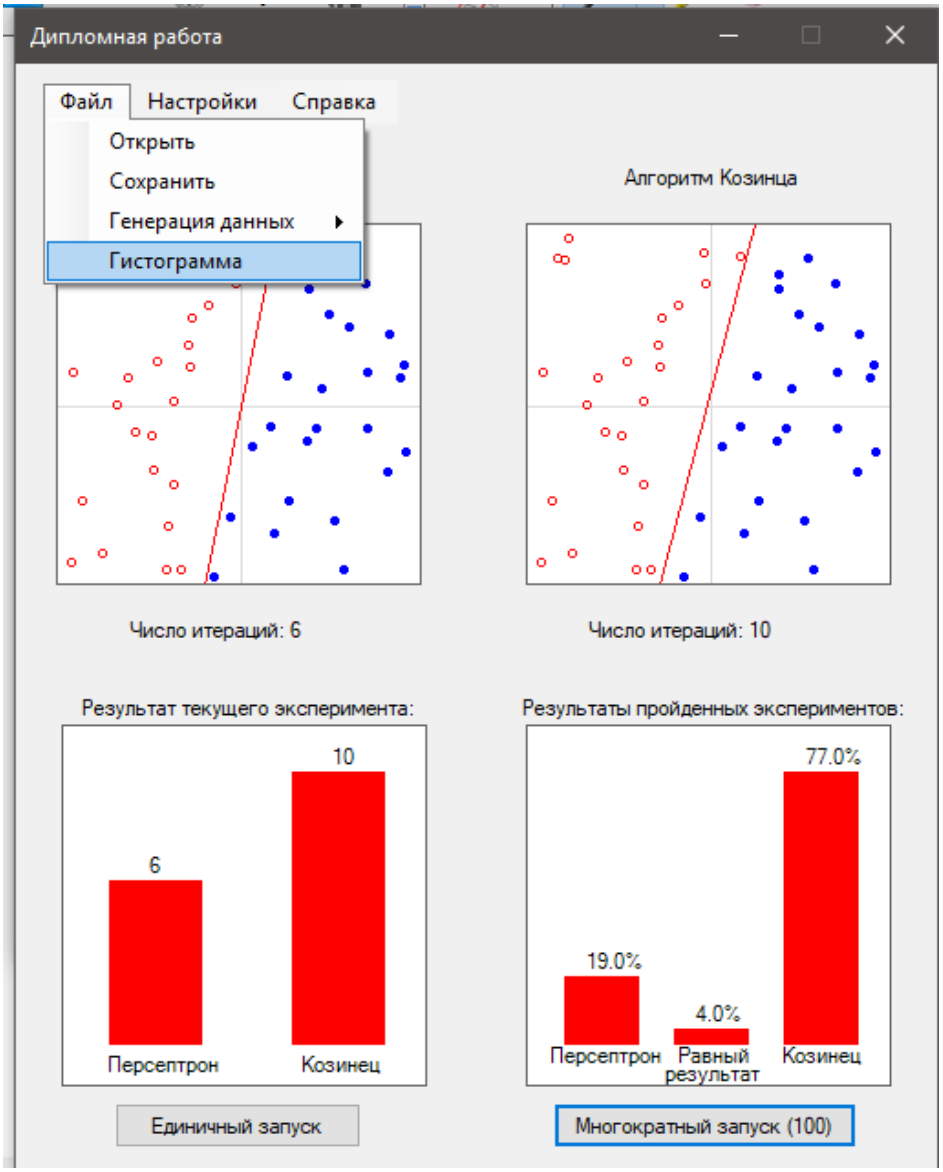


Рисунок 3.9. Відкриття вікна з гістограмою.

Після вибору цього пункту відкривається нове вікно (рис. 3.10), яке дозволяє відстежити кількість затрачених ітерацій на кожному з запусків, а також відображає:

- найменшу кількість ітерацій затрачених алгоритмом Козинця;
- найбільшу кількість ітерацій затрачених алгоритмом Розенблата;
- коефіцієнт варіації ітерацій під час множинних запусків алгоритму Козинця;
- найменшу кількість ітерацій затрачених алгоритмом Розенблата;
- найбільшу кількість ітерацій затрачених алгоритмом Розенблата;
- коефіцієнт варіації ітерацій під час множинних запусків алгоритму Розенблата.

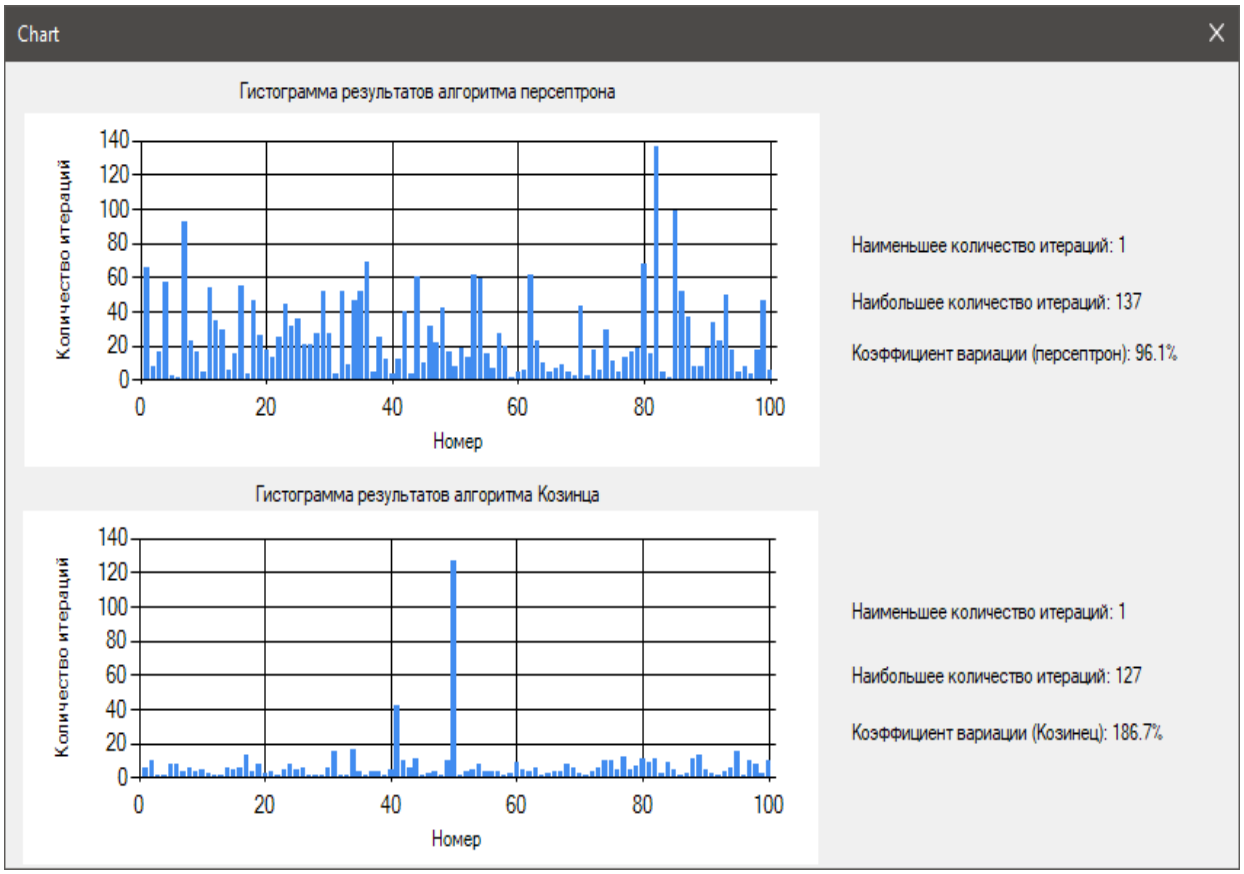


Рисунок 3.10. Вікно гістограми.

Висновки до розділу 3

В розділі детально описано всі можливі маніпуляції з програмою. Наведено знімки програми з поясненнями для кращого розуміння можливих дій. Розглянуто вибір даних для алгоритмів, запуск експерименту та налаштування програми.

РОЗДІЛ 4
РЕЗУЛЬТАТИ СТАТИСТИЧНИХ ЕКСПЕРИМЕНТІВ

Розглянемо результати, отримані при експериментальному дослідженні властивостей алгоритмів навчання Розенблата і Козинця на вибірках випадкових спостережень. Для наочності продемонструємо ці результати на прикладах лінійно-роздільних точок на площині.

Перша серія експериментів була спрямована на оцінку швидкості збіжності алгоритмів при різній кількості точок. На фіксованих числах точок $K = 10, 20, ..., 200$ випадково розташованих на площині, проводилися серії експериментів з навчання алгоритмів. У кожному окремому експерименті визначалися числа ітерацій $U_1(K)$ і $U_2(K)$ до зупинки відповідно алгоритму Розенблата і Козинця [48].

Порівняння чисел $U_1(K)$ і $U_2(K)$ дозволяло визначити лідера окремого експерименту, який розділив точки за менше число ітерацій. Далі визначався відсоток лідерства в серії з 5000 експериментів (рис. 2.3).

Експерименти показали, що приблизно в 20% випадків, при $K < 40$ спостережень, обидва алгоритми вимагали однакової кількості ітерацій. Зі збільшенням кількості точок алгоритм навчання Козинця опинявся абсолютним лідером (рис. 4.1).

У той же час, навіть при $K > 100$ в 10% випадків (в одному з десяти) алгоритм Розенблата навчався швидше ніж алгоритм Козинця [48].

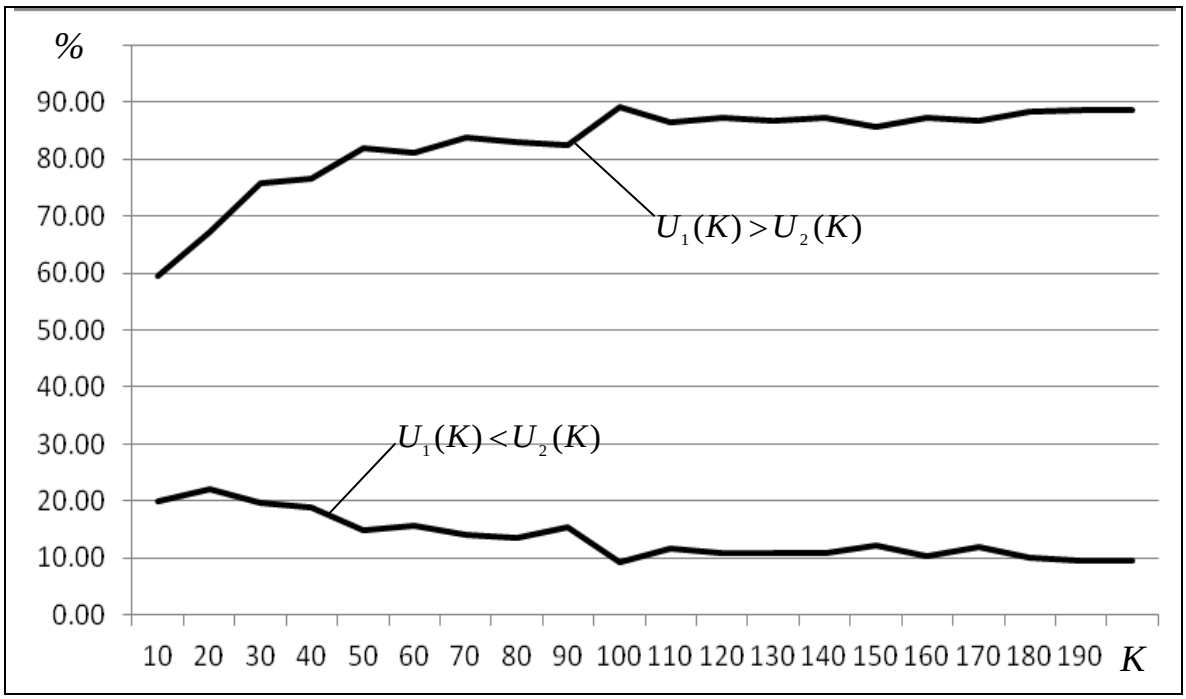


Рисунок 4.1. Графік залежності відсотка лідерства алгоритмів від кількості точок: $U_1(K) < U_2(K)$ – лідер алгоритм Розенблата;

$U_1(K) > U_2(K)$ – лідер алгоритм Козинця.

Експерименти також показали, що швидкість збіжності алгоритму Розенблатта залежить не тільки від числа оброблюваних точок, а й від їх взаємного розташування на площині.

Для ілюстрації цього факту розглянемо результати багаторазових експериментів навчання за двома вибірками, що складаються всього лише з трьох точок - двох точок класу V_1 і однієї точки класу V_2 (рис. 4.2).

Вибірка, показана на рис. 4.2, ліворуч виявилося «простою» для обох алгоритмів: для поділу класів алгоритм Розенблата вимагав 12 ітерацій, а алгоритм Козинця всього 1 ітерацію.

У той же час, при навчанні алгоритмів на вибірці, показаній на рис. 4.2, праворуч, швидкості збіжності алгоритмів істотно відрізнялися. Алгоритм Козинця справлявся із завданням всього лише за 2 ітерації, в той же для алгоритму Розенблата така вибірка виявлялася «важкою»: для поділу точок потрібно більше 500 ітерацій [48].

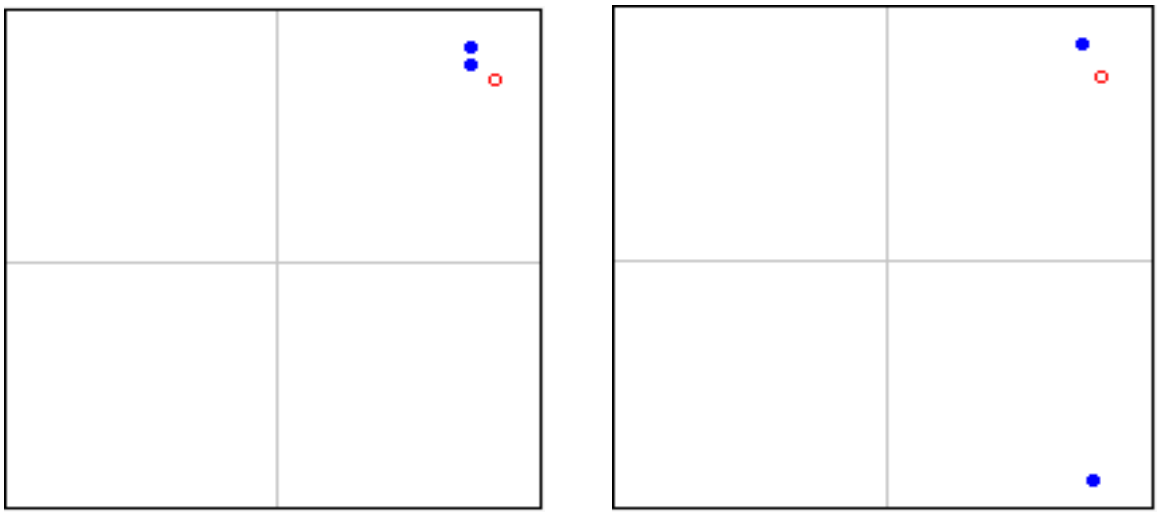


Рисунок 4.2. «Проста» (ліворуч) і «важка» (праворуч) навчальні вибірки.

Для пояснень цього ефекту розглянемо динаміку зміни положення дискримінантної функції в процесі навчання алгоритму Розенблата на вказаних двох вибірках.

Відповідно до виразу (2.7), для випадку $N = 2$ корекція положення лінійної дискримінантної функції

$$w_2 x_2 + w_1 x_1 + w_0, \tag{4.1}$$

яка відбувається при неправильній класифікації точки, зводиться до трьох операцій

$$w_2^{(t)} = w_2^{(t-1)} + \gamma \delta_\alpha^{(t)} x_2^{(t)},$$
$$w_1^{(t)} = w_1^{(t-1)} + \gamma \delta_\alpha^{(t)} x_1^{(t)},$$
$$w_0^{(t)} = w_0^{(t)} + \gamma \delta_\alpha^{(t)},$$

де $\delta_\alpha^{(t)}$ – помилка класифікації.

В процесі навчання на «простій» вибірці розділяюча пряма (4.1) від початкового положення (рис. 4.3, 1), що відповідає вектору $w^{(0)} = (0,0,1)$, поступово змінює свій напрямок (рис. 4.3, 2-5) і за порівняно невелику

кількість ітерацій приймає остаточне положення (рис. 4.3, 6), при якому точки навчальної вибірки правильно класифікуються. При цьому зміни положення прямої головним чином обумовлені зміною її нахилу, що визначається відношенням $w_1^{(t)} / w_2^{(t)}$, оскільки поділ точок «простої» вибірки практично не вимагає зміщення прямої щодо початку координат, тобто корекції відношення $w_0^{(t)} / w_2^{(t)}$.

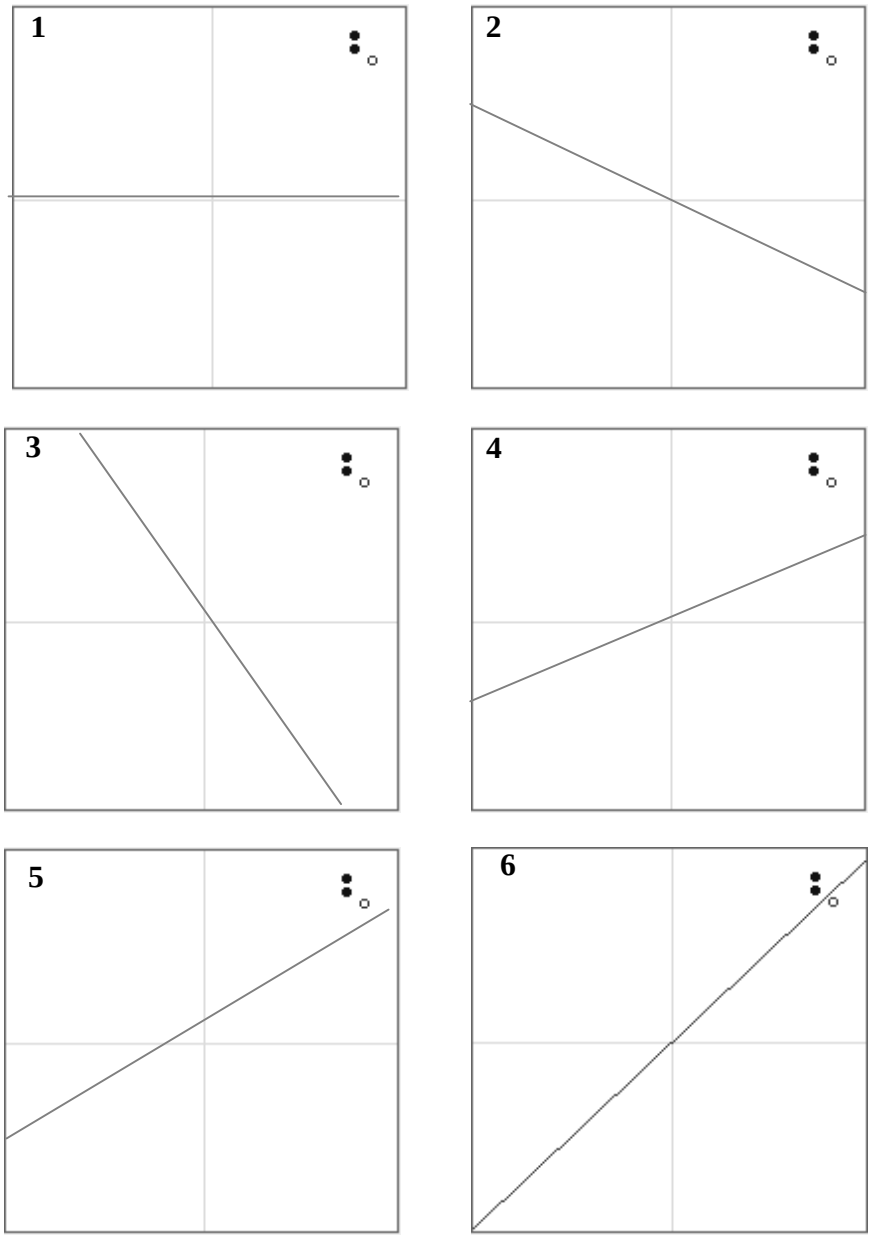


Рисунок 4.3. Етапи процесу навчання алгоритму Розенблата на «простій» вибірці.

При навчанні ж алгоритму Розенблатта по «важкій» вибірці динаміка зміни положення дискримінантної функції зовсім інша (рис. 4.4) [48].

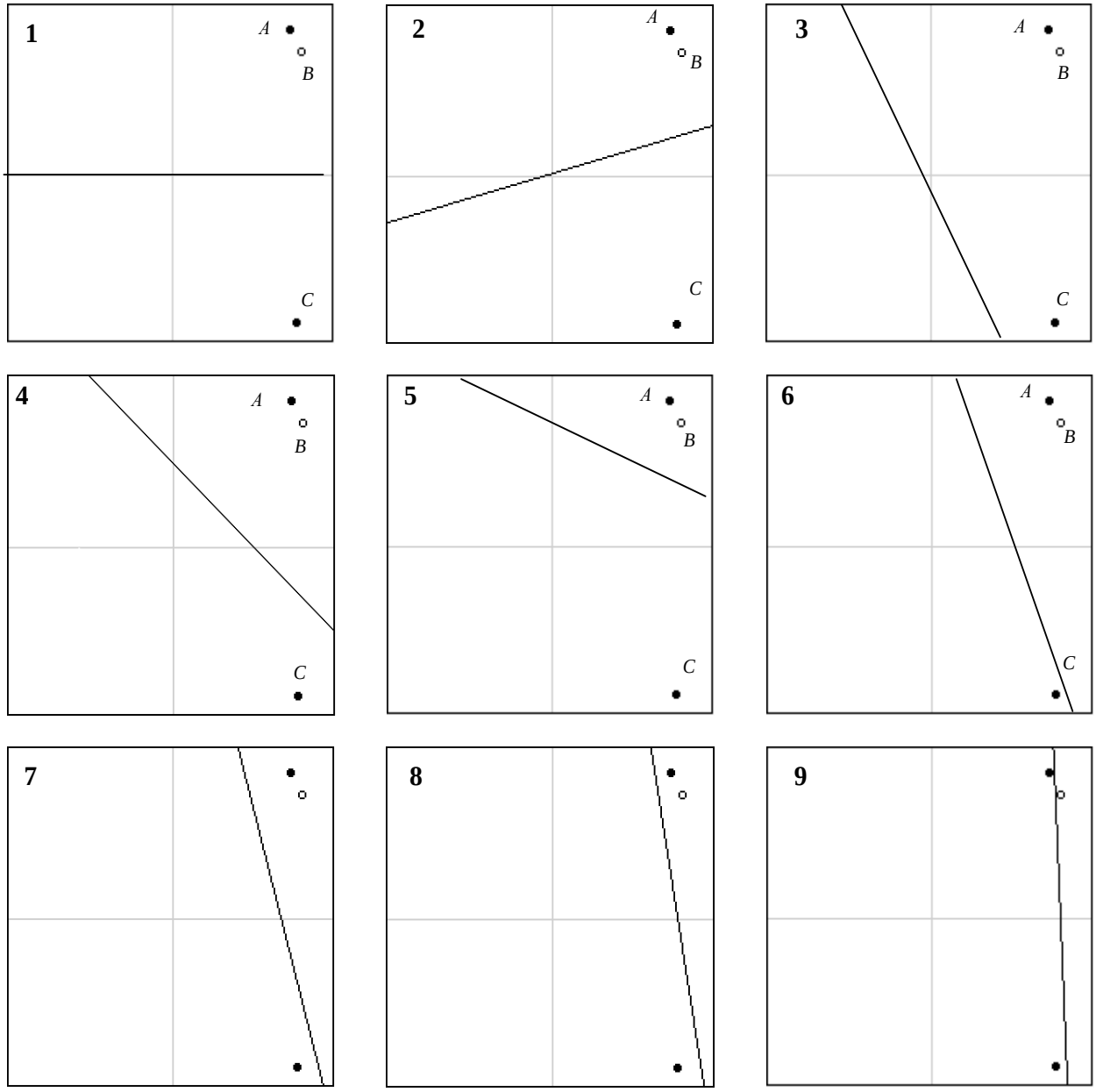


Рисунок 4.4. Етапи процесу навчання алгоритму Розенблата на «важкій» вибірці.

Відмітна особливість «важкою» вибірки від «простої» полягає в істотному розходженні відстаней по вертикалі $\Delta x_2 = \left| x_2^A - x_2^C \right|$ і по горизонталі $\Delta x_1 = \left| x_1^A - x_1^C \right|$ між точками A і C одного класу:

$$\Delta x_2 >> \Delta x_1 . \qquad (4.2)$$

Іншими словами, в даному випадку точки A і C одного класу істотно відрізняються за однією з координат і практично збігаються по іншій. Це призводить до того, що при коригуванні потрібно змінювати не тільки нахил дискримінантної функції $w_1^{(t)} / w_2^{(t)}$ (рис. 4.4, 1-3), але і її зміщення $w_0^{(t)} / w_2^{(t)}$ (рис. 4.4, 4-9).

Тому на «важкій» вибірці алгоритм Розенблата хоч і сходиться за кінцеву кількість кроків (хай живе теорема Новикова!), Але й число таких кроків досить велику.

Для ілюстрації на рис. 4.5 та рис. 4.6 показана динаміка зміни відношень $w_1^{(t)} / w_2^{(t)}$ і $w_0^{(t)} / w_2^{(t)}$ в процесі навчання на «простій» (вгорі) і «важкій» (внизу) вибірках. Легко помітити, що на «простій» вибірці нахил дискримінантної функції практично не змінювався, в той час як на «важкій» він змінювався від 0 до 200 у.о.

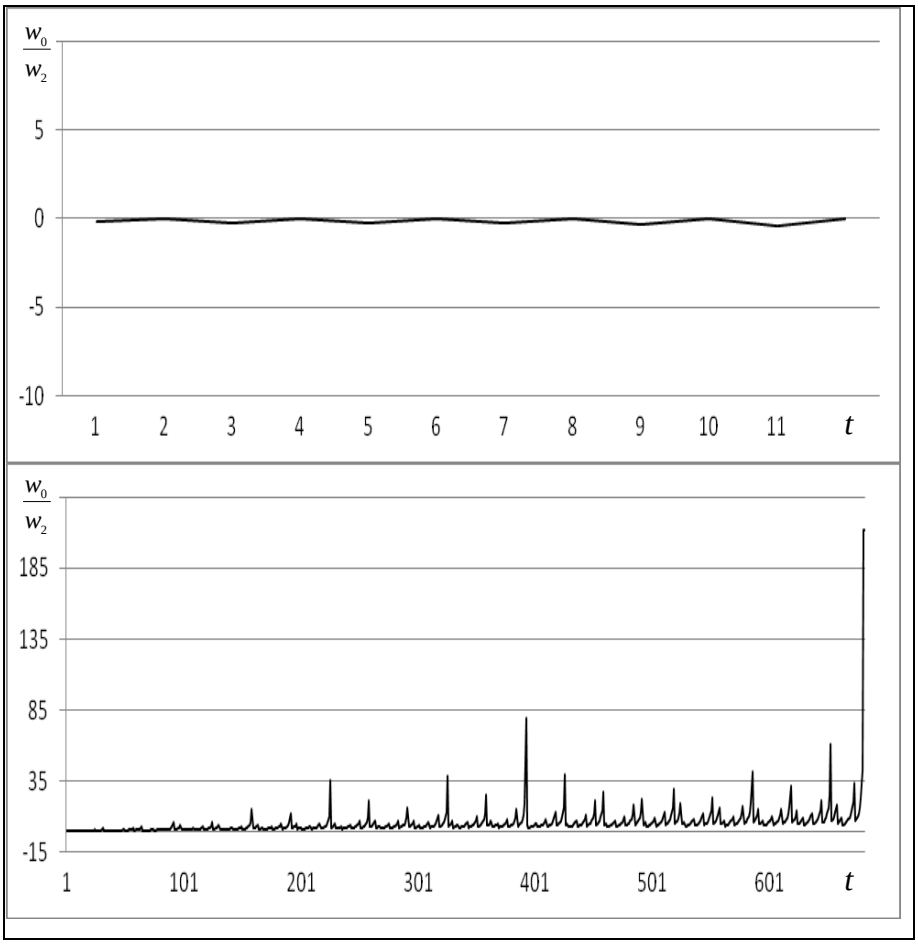


Рисунок 4.5. Динаміка зміни зсуву лінійної дискримінантної функції на «простій» (вгорі) і «важкій» (внизу) вибірках.

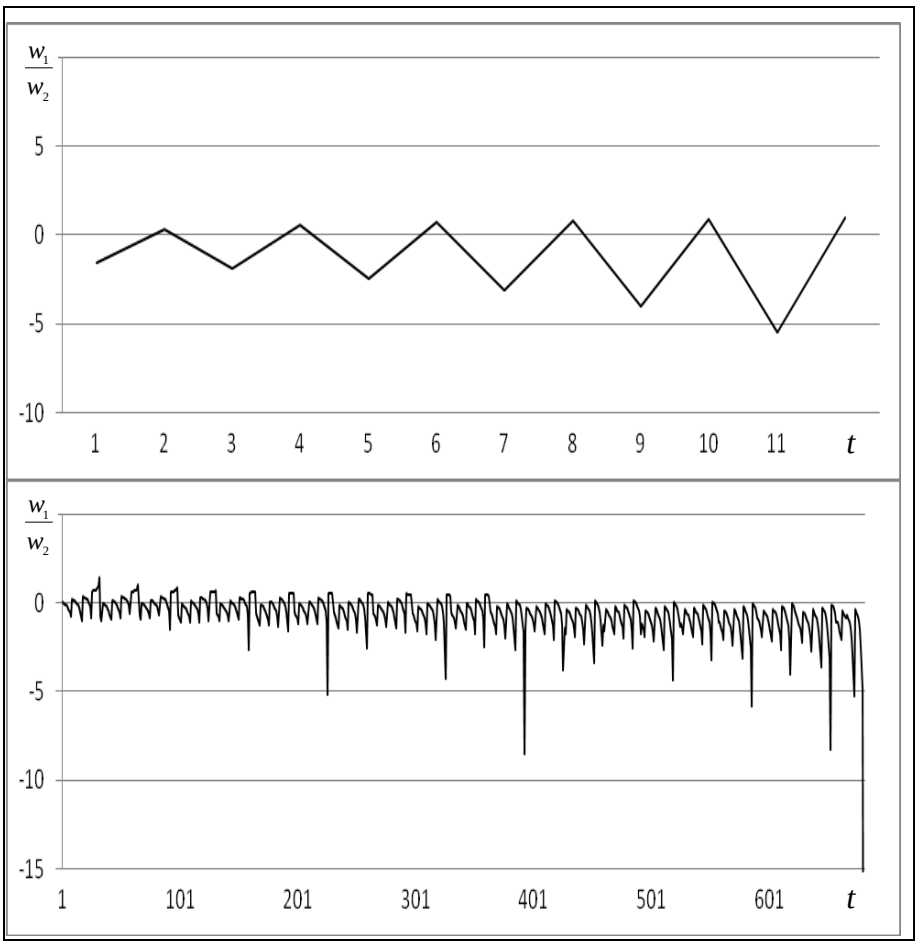


Рисунок 4.6. Динаміка зміни нахилу лінійної дискримінантної функції на «простій» (вгорі) і «важкій» (внизу) вибірках.

Слід звернути увагу на одну важливу відмінність процедур навчання алгоритмів Розенблатта і Козинця. Як уже зазначалося, алгоритм Козинця передбачає пошук першої неправильно класифікованої точки вибірки, яка призводить до чергової корекції вектора параметрів дискримінантної функції [48].

Оскільки пошук таких точок здійснюється випадковим чином, то для прискорення процесу навчання доцільно на кожному черговому етапі перегляду точок навчальної вибірки виключати можливість багаторазової перевірки виконання умови $\langle w, x_k \rangle \cdot c_k > 0$ правильної класифікації для одних і тих же спостережень x_k, $k \in [1, n]$.

З цією метою була розроблена нескладна оптимізаційна процедура, що забезпечує формування скороченого підмножини спостережень для

чергового кроку корекції вектора параметрів. Зрозуміло, після завершення цього кроку пошук нової неправильно класифікованої точки проводиться по всій початковій вибірці спостережень.

Статистичні експерименти показали, що в середньому використання оптимізаційної процедури дозволяє прискорити час збіжності алгоритму Козинця більш ніж в 8 разів.

При виконанні досліджень проводився аналіз варіації кількості витрачених ітерацій кожного з алгоритмів навчання на вибірках різного об'єму. Число необхідних ітерацій при багаторазовому навчанні алгоритму на конкретній вибірці розглядалося як реалізації випадкової величини, для якої визначався коефіцієнт варіації Пірсона [12] – відношення середньоквадратичного відхилення числа ітерацій до середнього значення ітерацій, виражений у відсотках.

Експерименти показали, що, незважаючи на те, що зі збільшенням обсягу вибірки алгоритм Козинця вимагає значно більшого числа ітерацій в порівнянні з алгоритмом Розенблата (рис. 4.7).

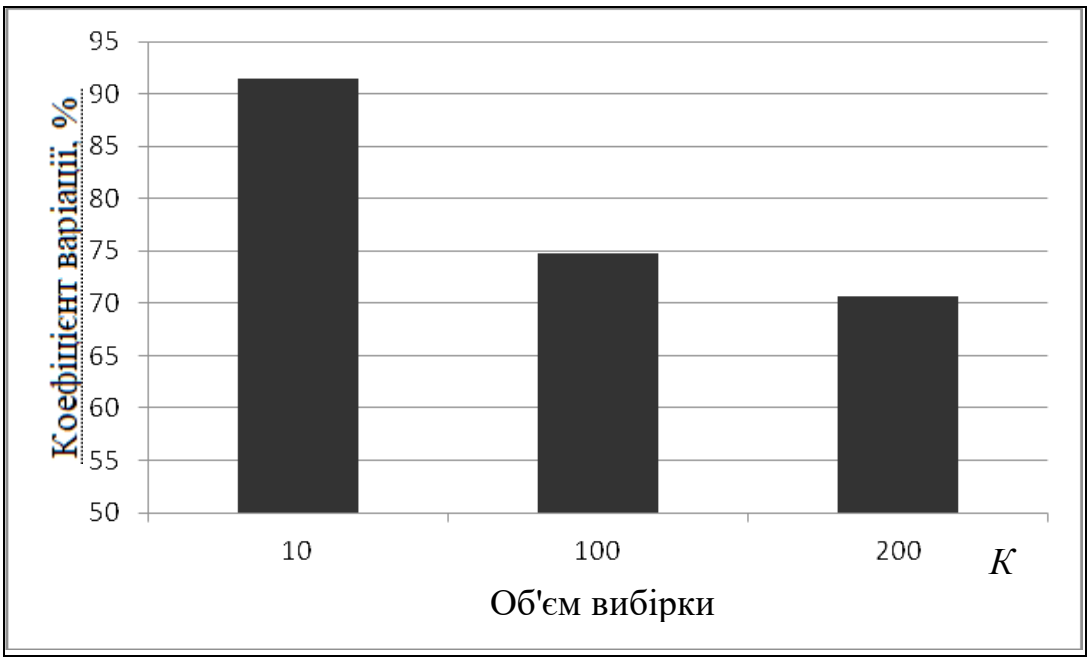


Рисунок 4.7. Залежність коефіцієнта варіації числа ітерацій алгоритму Козинця від обсягу вибірки.

З ростом числа точок у вибірці відбувається зменшення коефіцієнту варіації числа ітерацій. При збільшенні обсягу вибірки від 10 до 200 точок коефіцієнт варіації числа ітерацій зменшився на 20%.

Нагадаємо, що алгоритм навчання Розенблатта пропонує принцип налаштування параметрів лінійної дискримінантної функції (2.3), яка структурно реалізує схему персептрона (рис. 4.8) - базового елементу нейронних мереж [21], активно застосовуються для вирішення багатьох прикладних задач [22].

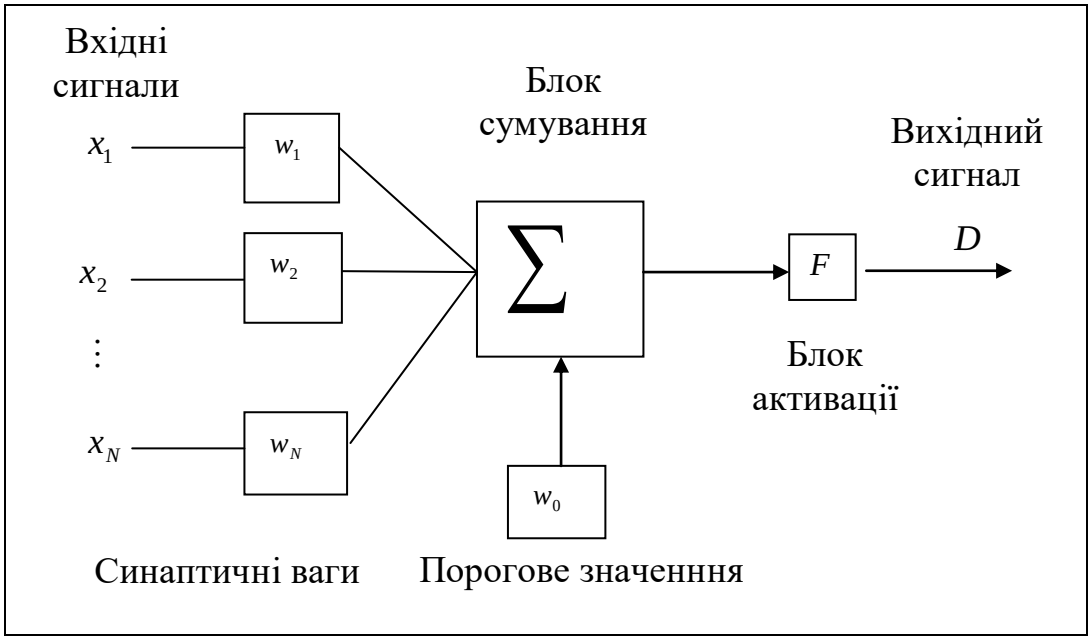


Рисунок 4.8. Одношаровий персептрон Розенблата.

Зрозуміло, що алгоритм Козинця, по суті, пропонує альтернативний підхід до навчання цієї ж схеми. Проведені нами експериментальні дослідження, які підтвердили високу швидкість збіжності алгоритму Козинця, дозволяють сподіватися, що використання цього алгоритму в якості алгоритму навчання базових елементів нейронної мережі дозволить підвищити їх ефективність. Врешті-решт, вивчення такої можливості є перспективним [48].

Нагадаємо, що для характеристик процесу навчання нейронних мереж введений спеціальний термін - «епоха навчання» [13], під яким

розуміють етап корекції параметрів дискримінантної функції при одноразовому перегляді всіх точок навчальної вибірки.

Зрозуміло, що для алгоритму навчання Козинця «епоха навчання» і «крок ітерації» еквівалентні поняття. У той же час для алгоритму Розенблатта «епоха навчання» складається з послідовності корекцій параметрів розділяючої функції при одноразовому перегляді всієї вибірки і виявленні кожного невірно класифікованого спостереження.

Беручи до уваги зазначені тлумачення термінів, в експериментах проводився порівняльний аналіз кількості «епох навчання», що витрачаються одним і іншим алгоритмів на серії випадково породжуваних спостережень.

Експерименти показали (рис. 4.9), що при числі точок $K < 70$ перевагу має алгоритм навчання Козинця, а в міру збільшення числа K зростає відсоток лідерства алгоритму навчання Розенблата [48].

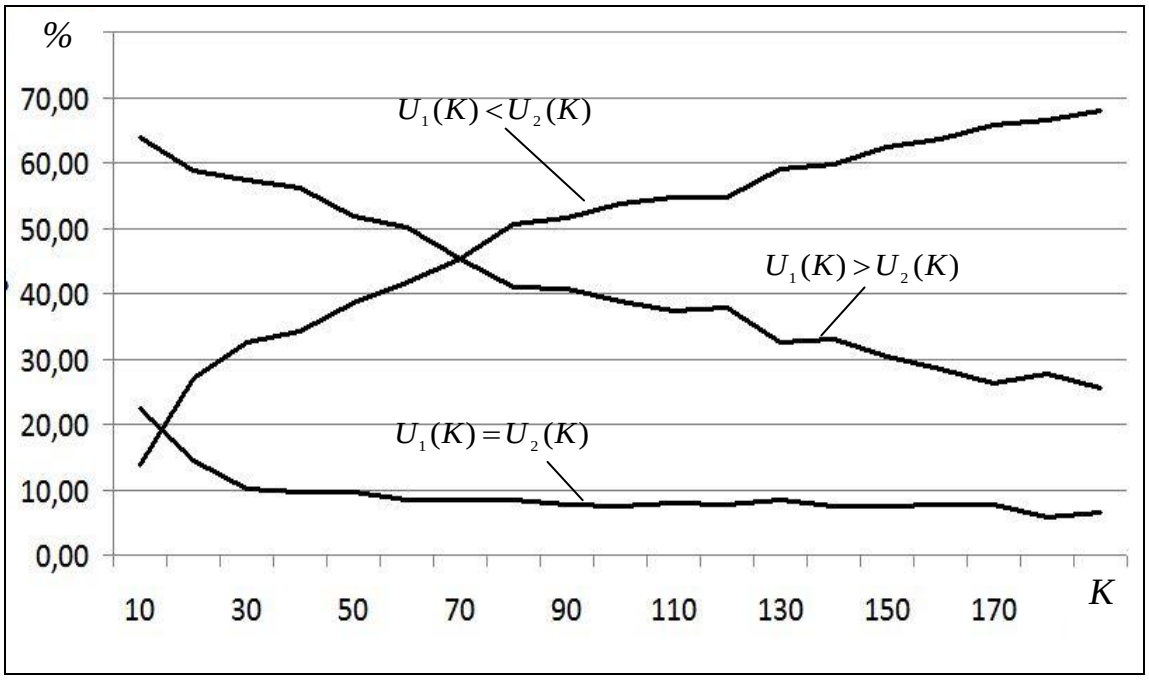


Рисунок 4.9. Графік залежності кількості «епох навчання» алгоритмів від кількості точок: $U_1(K) < U_2(K)$ - лідер алгоритм Розенблата; $U_1(K) = U_2(K)$ - лідер відсутній; $U_1(K) > U_2(K)$ - лідер алгоритм Козинця.

Цікаво зауважити, що при $K > 30$ відсоток випадків, коли явний лідер був відсутній, тобто виконувалася умова $U_1(K) = U_2(K)$, практично не змінювався і не перевищував 10%.

Для ілюстрації на рис. 4.10 представлені приклади результатів навчання алгоритмів і відповідні гістограми лідерства.

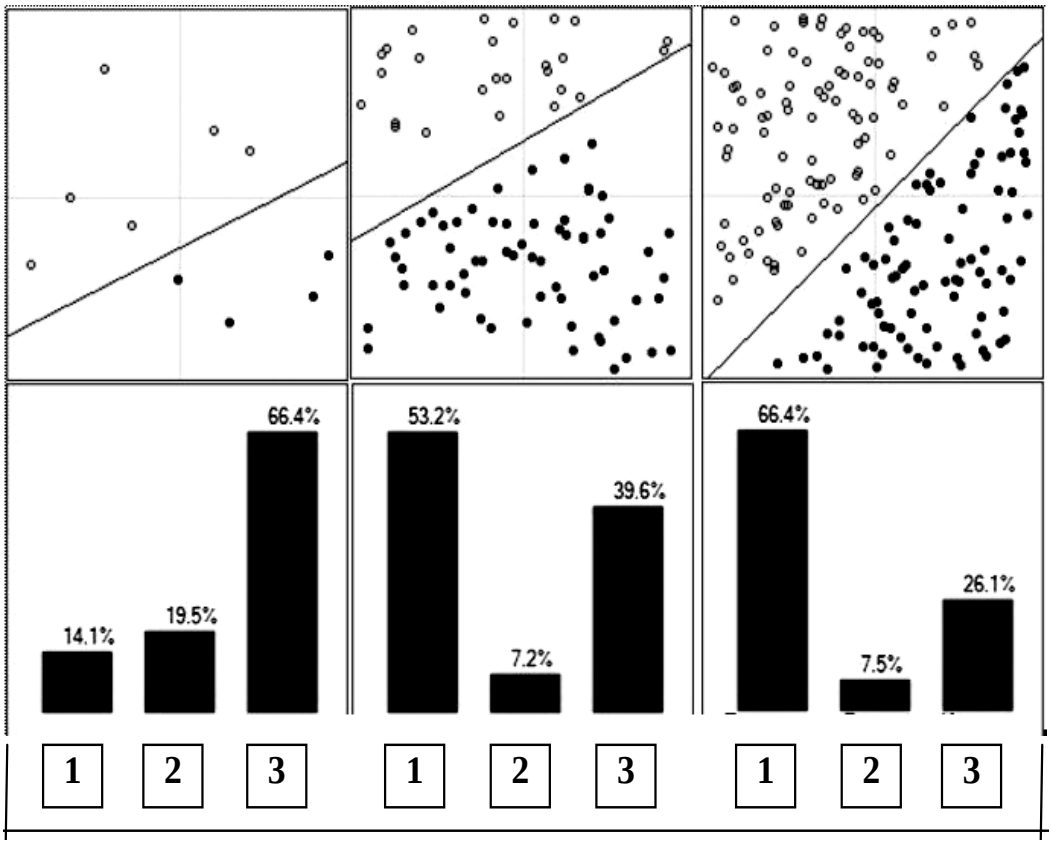


Рисунок 4.10. Приклади результатів навчання алгоритмів
1 - лідер алгоритм Розенблата; 2 - лідер від сутній; 3 - лідер алгоритм Козинця.

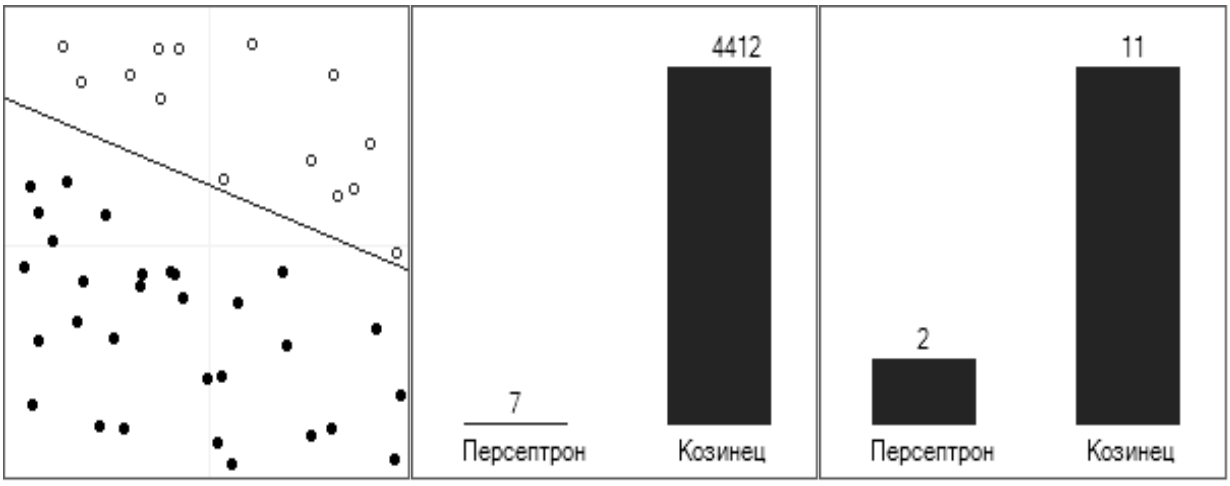
На початковому етапі експериментів був виявлений великий розкид результатів навчання алгоритму Козинця (рис. 2.5): на одних і тих же даних для навчання алгоритму потрібно від 2-х до 4000 ітерацій. Аналіз вихідної версії алгоритму, описаного в роботі [14] дозволив встановити причину таких розкидів.

Як випливає з опису алгоритму, на кожній ітерації здійснюється пошук точки, котра неправильно класифікується поточним становищем розділяючої гіперплощини. Оскільки алгоритм Козинця передбачає, що

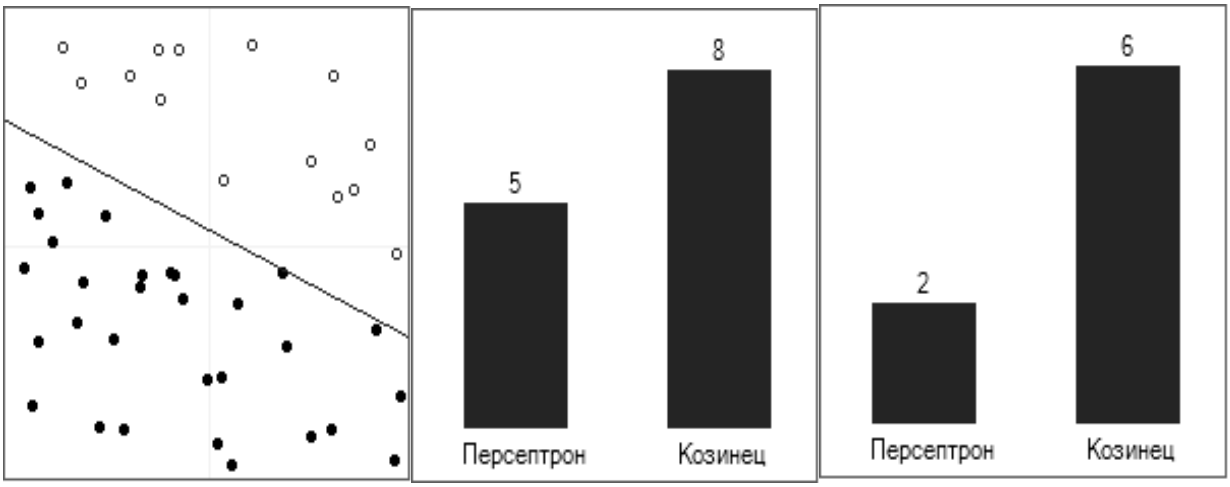
пошук невірно класифікованої точки здійснюється випадковим чином, то одна і та ж точка може бути обрана багаторазово, що призводить до збільшення числа ітерацій.

Нами запропонована невелика модифікація, котра передбачає, що на кожній ітерації після перевірки випадково обраної точки вона видаляється з вибірки [48].

Експерименти на серії випадково згенерованих даних підтвердили ефективність такої модифікації: модифікований алгоритм навчання дозволяє надійно визначити лінійну дискримінантну функцію без різких стрибків в кількості ітерацій (рис. 4.11).



a



b

Рисунок 4.11. Приклади роботи вихідного (a) і модифікованого (b) алгоритмів навчання Козинця на одній і тій же вибірці.

І все ж швидкість збіжності алгоритмів навчання головним чином характеризує число корекцій параметрів дискримінантної функції, а значить, з цієї точки зору, безперечним лідером можна вважати алгоритм Козинця, що ілюструє рис. 4.1.

Таким чином, розроблена програмна система дозволила на основі серій статистичних експериментів встановити раніше невідомі властивості алгоритмів навчання Розенблата і Козинця, провести їх порівняльний аналіз і намітити перспективи подальших досліджень щодо вдосконалення нейронних мереж [48].

Висновки до розділу 4

В розділі детально описано всі проведені експерименти, що дозволили виявити раніше невідомі властивості алгоритмів.

Статистичні експерименти, проведені з використанням розробленої про-програмних системи, показали, що при малому обсязі вибірки приблизно в 20% випадків швидкості збіжності алгоритмів Розенблатта і Козинця однакові. Зі збільшенням кількості спостережень алгоритм навчання Козинця опинявся абсолютним лідером і при числі спостережень $K > 100$ в 90% випадків навчався швидше, ніж алгоритм Розенблатта.

Швидкість збіжності алгоритму навчання Козинця менш чутлива до розміщенню точок в навчальній вибірці і з ростом числа спостережень коефіцієнта варіації число ітерацій зменшується.

РОЗДІЛ 5
СТАРТАП-ПРОЕКТ

Програмна система дозволяє обирати кращий алгоритм для лінійної класифікації серед запропонованих, що дозволяє суттєво зекономити час при вирішенні подібних проблем.

Складові:

1) підсистема зчитування даних;

2) підсистема обробки даних;

3) підсистема розрахунку алгоритму;

4) підсистема побудови площини поділу виведення результату.

Наявна можливість оцінки якості та швидкості різних алгоритмів класифікації для різних задач.

Бізнес-модель: орієнтація на дослідні інститути.

Цінний продукт.

1. Сукупність товарів-послуг (покращення товару-послуг).

Комплекс послуг: оцінка якості та швидкості різних алгоритмів класифікації для різних задач за рахунок використання різних алгоритмів класифікації та їх порівняльного аналізу.

2. Вирішує такі проблеми клініки:

- вибір найкращого алгоритму кластеризації для вибірки;
- побудова порівняльної моделі.

3. Підтримка і сервіс програмної системи.

Додатковий сервіс надається шляхом підтримки системи розробником.

4. Продуктова лінійка (унікальність).

Сукупність різних алгоритмів класифікації.

Унікальність пропозиції (новизна, продуктивність, дизайн, ціна, економія на витратах, зниження ризику, доступність, зручність).

На ринку не існує аналогів у даної програмної системи.

Сегмент споживачів

1. Ринок.

Ринок: середнє сегментування – надання послуг дослідним інститутам.

В майбутньому планується розширення списку сегментів (міжнародний ринок).

Канали збуту.

1. Прямі – прямий продаж через сайт.

Фріміум – частина послуг (наприклад, інформаційна) безкоштовна, частина – платна (або Shareware - 1 місяць безкоштовне надання послуг, далі – платно).

2. Функції .

Продажні (велике рішення) – в системі присутні всі модулі.

Надання рекомендацій в реальному часі – модуль вартістю 1000 грн.

Взаємодія із споживачами:

- залучення закладів, їх утримання. Заклад купує систему.
- підтримка (пошта, телефон, особисто, форум). Наприклад, на форумі можна повідомляти про внесення в систему вдосконалень (при цьому необхідні повторні продажі).
- залучення клієнтів завдяки науковим та медичним конференціям;
- повторний продаж системи завдяки реалізації додаткових модулів-послуг.

Дохід (монетизація).

- За що і скільки готовий платити клієнт? За систему, яка буде працювати та підтримуватися розробником.

Ключові види діяльності.

1. Процес створення цінності.

Адміністрування, розробка системи (в тому числі і наукова діяльність), навчання користувачів системи, створення додаткових модулів (за потребою)

2. Виробництво товару-послуг, продаж.

3. Підтримка рішення.

Ключові ресурси:

1. Матеріальні (в т.ч. комп'ютерна техніка) - 3 персональні комп'ютери, принтер.

2. Інтелектуальні (в т.ч. патенти та ліцензії).

3. Людські:

- бухгалтер веде всю фінансову діяльність фірми (нарахування і сплата податків, розподіл прибутку, розрахунок і видача зарплати);

- старший програміст здійснює розробку програм, програмно-технічних засобів і контролює їх якісне виконання. Під його контролем працюють два фахівці в даній області, які й реалізують успішне виконання проекту;

- маркетолог – за дослідження потреб, доцільності подальшого розширення системи;

- генеральний директор займається кадрами, укладає договори на поставку продукції в організації та установи, відвідує виставки, конференції з обміну досвідом, відповідає за поставку обладнання у випадку його зносу, технічного старіння. Забезпечує регулярну поставку сировини, проводить дослідження ринку, виконує розрахунки, пов'язані зі змінами в технології.

4. Фінансові - фінансування всіх членів команди (в т.ч., програмістів-розробників).

Ключові партнери:

1. Забезпечення ресурсами (комп'ютерною технікою та обладнанням).

2. Оптимізація та економія в продажах.

3. Зниження ризиків і невизначеності.

4. Подальший розвиток проекту.

Витрати

Таблиця 5.1

Витрати на паливо й енергію на технологічні цілі

Назва	Кількість	Потужність обладнання, кВт	Корисний фонд часу, годин	Коефіцієнт за часом	Коефіцієнт за потужністю	Тарифи за одну кВт/год. енергії, грн.
Комп'ютер	4	0,40	0,01	0,50	0,50	140,70
Всього	4	0,40	0,01	0,50	0,50	140,70

Таблиця 5.2

Розрахунок собівартості одиниці продукції

Найменування статей калькуляції	Всього, грн.	Питома вага, %
Попутні комплектуючі вироби і напівфабрикати	166,4500	76,4470
Паливо й енергія на технологічні цілі	1,9698	0,9047
Основна заробітна плата	11,0000	5,0521
Додаткова заробітна плата	3,3000	1,5156
Відрахування на соціальне страхування	5,3768	2,4695
Відшкодування зносу спеціальних інструментів	0,0019	0,0009
Витрати на утримання і експлуатацію обладнання	0,1670	0,0767

Продовж. табл. 5.2

Найменування статей калькуляції	Всього, грн.	Питома вага, %
Загальновиробничі витрати	12,2837	5,6416
Виробнича собівартість	200,5492	92,1080
Адміністративні витрати	11,1670	5,1288
Інші витрати	2,0055	0,9211
Витрати на збут	4,0110	1,8422
Повна собівартість	217,7327	100,0000

Планується продаж 2000 ліцензій в місяць, вартість яких становить 435465,3726 грн.

Фінансовий план.

Витрати.

Таблиця 5.3

Витрати на місяць

Статті витрат	Сума, грн
1. Постійні витрати	62370,21
1.1 Орендна плата	7680,00
1.2 Заробітна плата працівникам	33000,00
1.3 Нарахування на заробітну плату	17028,00
1.4 Плата за телефон та інтернет	119,81
1.7 Витрати на електроенергію	3939,60
1.8 Амортизаційні відрахування	602,80
2. Змінні витрати (на перший місяць)	396834,35
2.1. Витрати на закупку матеріалів та техніки	391834,35
2.2. Витрати на рекламу	4000,00

Продовж. табл. 5.3

Статті витрат	Сума, грн
2.3. Інші невраховані витрати	1000,00
Вартість устаткування	40000,00
Всього	499204,56

Постійні витрати:

- орендна плата - 7680 грн/міс;
- заробітна плата працівникам;
- генеральний директор – 5000 грн;
- бухгалтер – 4000 грн;
- старший програміст – 4000 грн;
- старший інженер – 4000 грн;
- програміст – 3500 грн;
- прибиральниця – 2000 грн;

Всього: 33000 грн.

Відрахування з заробітної плати:

- до пенсійного фонду – 35,2%;
- єдиний соціальний внесок – 16,4%;

Всього: 51,6% (51,6% від 33000,00грн = 17 028 грн).

Амортизаційні відрахування:

- зношення обладнання становить 10% від балансової вартості обладнання на рік – 4000 грн., а в місяць – 334 грн;
- зношення споруди 3.5% від балансової вартості на рік – 3225,6 грн, а в місяць - 268, 8 грн;

Визначення місячної виручки.

Пристрій ми плануємо продавати в офісі за ціною 300 грн за 1 шт (300, 00 • 2000 = 600000 грн).

Розрахунок точки беззбитковості

Точка беззбитковості дозволить визначити, коли проект перестане бути збитковим (рис. 5.1).

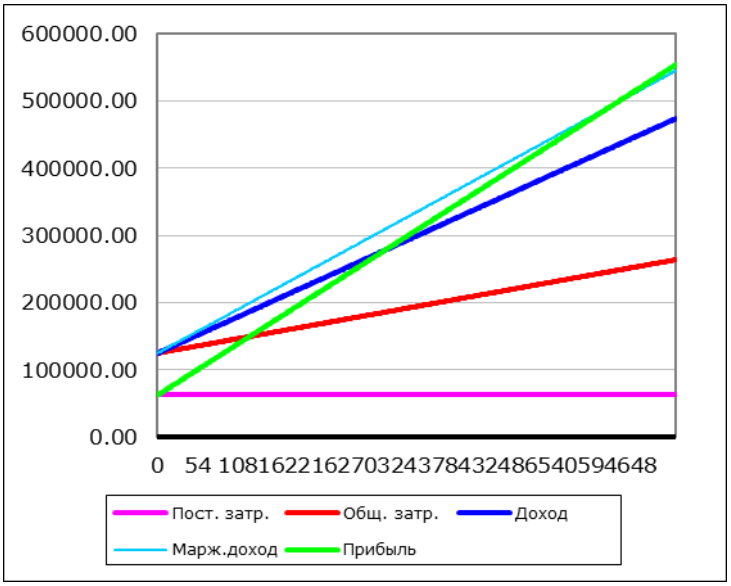


Рисунок 5.1. Точка беззбитковості.

Вийшло, що величина точки беззбитковості дорівнює 613,98 штук, тобто необхідно випустити 614 ліцензій, і тільки після цього підприємство стане отримувати прибуток.

Окупність

Рисунок 5.2. Період окупності.

Дисконтований період окупності 2,6084 місяці.

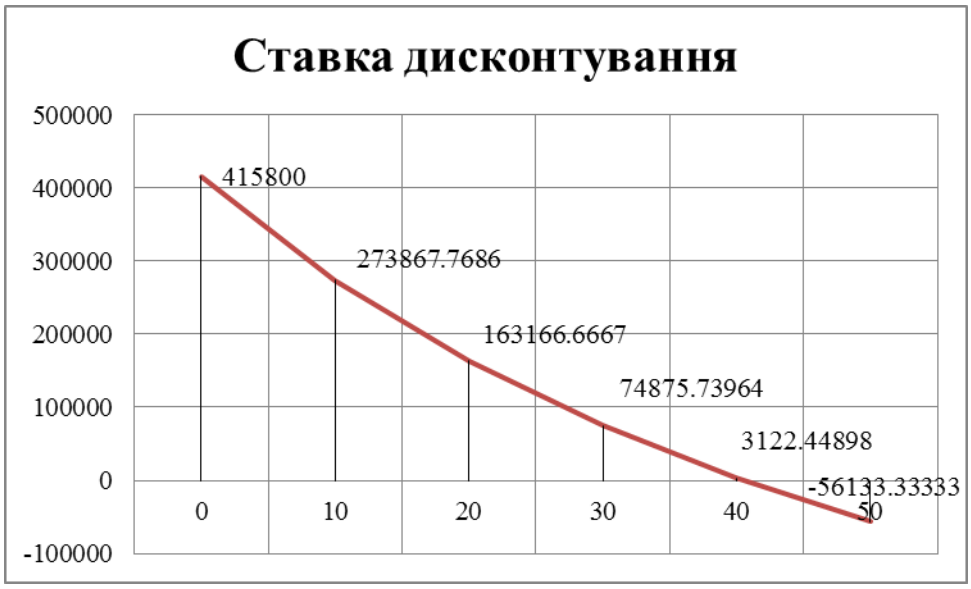


Рисунок 5.3. Ставка дисконтування.

IRR ≈ 41 % (ставка рентабельності за якої чиста приведена вартість 0).

Споживчі властивості товару:

1. Базові (очікувані) властивості:

- придбавши систему клініка очікує на підвищення відсотку вірно встановлених діагнозів.

2. Основні (бажані) властивості:

- зручність під час використання;
- точність результату;
- надійність у роботі;

Дослідження ринку.

Під час початкового періоду дослідження ринку потрібно відвідувати медичні конференції, на яких розглядаються дані проблеми.

Дослідження конкурентного оточення.

Повний доступ до всіх ресурсів можна здійснити одразу після встановлення додатку на комп’ютер (тобто процедура реєстрації не потрібна). Користувачам видається повний доступ до інформації зі змогою редагування даних.

Конкурентів у програмної системи немає.

План розвитку товару:

- надійність товару в споживанні: зведення помилок програми до мінімуму;
- ергономічні властивості: зручність експлуатації товару - зручний інтерфейс системи;
- естетичні властивості: здатність товару виражати свою соціокультурну значимість, ступінь корисності та досконаліть;
- безпека споживання: система є безпечною при використанні;
- як зміна характеристик продукту змінює споживчі властивості: додавання модулів до системи розширює споживчі властивості. Збільшення функцій (модулів) збільшує ціну товару.

Додаток використовують в комплексі із комп'ютером.

Управління ціною:

1. Формування ціни на продукт (витрати на розробку системи, організаційні заходи, рекламні заходи).

2. Формування системи лояльності: системи знижок, заохочень (для збільшення продажів, повторних продажів, партнерських продажів): робота із БД клінік для покращення алгоритмів.

Висновки до розділу 5

Створено стартап-проект на основі магістерської дисертації. Розраховано його фінансовий плвн, точку беззбитковості. Створено план розвитку товару. Розраховано собівартість одиниці товару.

ВИСНОВКИ

1. В результаті виконання магістерської дисертації було проведено аналіз існуючих алгоритмів лінійної класифікації об’єктів.

2. За допомогою контекстної діаграми, моделі варіантів використання, діаграми станів, діаграми послідовності, діаграми кооперації, діаграми діяльності було проведено детальне проектування, що реалізує створені моделі.

3. Розроблено програмний продукт за допомогою мови програмування C# в середовищі Visual Strudio 2017. Він являє собою зручну програму для дослідження властивостей алгоритмів Розенблата и Козинця.

4. Статистичні експерименти, проведені з використанням розробленої про-програмних системи, показали, що при малому обсязі вибірки приблизно в 20% випадків швидкості збіжності алгоритмів Розенблатта і Козинця однакові. Зі збільшенням кількості спостережень алгоритм навчання Козинця опинявся абсолютним лідером і при числі спостережень $K > 100$ в 90% випадків навчався швидше, ніж алгоритм Розенблатта.

5. Швидкість збіжності алгоритму навчання Козинця менш чутлива до розташування точок в навчальній вибірці і з ростом числа спостережень коефіцієнта варіації число ітерацій зменшується.

6. Більш висока швидкість збіжності алгоритму Козинця в порівнянні з алгоритмом Розенблатта, підтверджена серіями проведених статистичних експериментів, дозволяє сформулювати перспективний напрямок досліджень по розвитку нейронних мереж, в яких алгоритм Козинця буде використаний для настройки базових елементів - персептронів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hastie T. The elements of statistical learning [Text] / T. Hastie, R. Tibshirani, J. Friedman. – New York: Springer, 2014. – 739 p.

2. Bishop C. M. Pattern recognition and machine learning [Text] / C. M . Bishop. – New York: Springer, 2006. – 738 p.

3. Merkov A.B. Image recognition: Introduction to statistical learning methods [Text] / A.B . Merkov. – Moscow: URSS, 2011. – 256 p.

4. Vapnik V. The nature of statistical learning theory [Text] / V. Vapnik. – New York: Springer-Verlag, 1995. – 188 p.

5. Gori M. Machine Learning: A constraint-based approach [Text] / M. Gori. – Waltham: Morgan Kaufmann; 2017. – 580 p.

6. Штучні нейронні мережі [Електронний ресурс] – Режим доступу до ресурсу: http://gymlit.in.ua/re_Штучні%20нейронні%20мережі.

7. Харарі Ю.Н. Людина розумна. Історія людства від минулого до майбутнього [Текст] / Ю.Н. Харарі. – Family Leisure Club, 2016. – 413 с.

8. Терехов С.А. Лекції з теорії і додатків штучних нейронних мереж. Лабораторія штучних нейронних мереж НТО – 2 [Текст] / С.А. Терехов. – Снежинск – ВНДІТФ, 2010. – 104 с.

9. Kodratoff Y. Machine learning: an artificial intelligence approach [Text] / Y. Kodratoff, R. S. Michalski., Vol. 3. – Moskow: Elsevier; 2014. – 825 p.

10. Camastra F. Machine learning for audio, image and video analysis: Theory and Applications [Text] / F. Camastra, A. Vinciarelli. – Madrid: Springer; 2015. – 561 p.

11. Schlesinger M. Ten lectures on statistical and structural pattern recognition [Text] / M. Schlesinger, V. Hlavac. – Dodrecht/Boston /London: Kluwer Academic Publishers; 2002. – 519 p.

12. Burstein F. Handbook on decision support systems 2: variations [Text] / F. Burstein, Cl. W. Holsapple. – New York: Springer; 2008. – 798 p.

13. Kung S. Y. Kernel methods and machine learning [Text] / S. Y. Kung. – Cambridge: Cambridge University Press; 2014. – 591 p.

14. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain [Text] / F. Rosenblatt // Psychological Review. – 1958. – № 65(6), – P. 386-408.

15. Kozinets B.N. A recursive algorithm for dividing the convex hulls of two sets [Text] / B.N. Kozinets // Computer Science and Programming. – 1966. – № 4. – P. 43-50.

16. Novikoff A. B. On convergence proofs on perceptrons [Text] / A. B. Novikoff // Symposium on the Mathematical Theory of Automata. – 1962. – № 12, – P. 615-622.

17. Vapnik V.N. Theory of pattern recognision [Text] / V.N. Vapnik, A.J. Chervonenkis. – Moscow: Nauka; 1974. – 416 p.

18. Buslenko N.P. The method of statistical modeling [Text] / N.P. Buslenko. – Moscow: Statistics; 1970. – 113 p.

19. Rubinstein R. Y. Simulation and the Monte Carlo method (3 ed.) [Text] / R. Y. Rubinstein, D. P. Kroese. – New York: John Wiley & Sons; 2016. – 432 p.

20. Robert C. P. Monte Carlo statistical methods (2nd ed.) [Text] / C. P. Robert, G. Casella. – New York: Springer; 2004. – 649 p.

21. Trickey K. A. Structural models of coefficient of variation matrices [Text] / K. A. Trickey. – Los Angeles: University of California; 2015. – 233 p.

22. Kussul E. Neural networks and micromechanics [Text] / E. Kussul, T. Baidyk, D.C. Wunsch. – New York: Springer Science & Business Media, 2009. – 221 p.

23. Misuno I.S. Experimental investigation of handwritten digit classification [Text] / I.S. Misuno, D.A. Rachkovskij, S.V. Slipchenko // System Technologies. – 2005. – № 4(39). – P. 110-133.

24. Short term load forecasting with multilayer perceptron and recurrent neural networks [Text] / Muhammad Riaz Khan, Cestm Ondrusek // Journal of ELECTRICAL ENGINEERING, VOL. 53, NO. 1-2, 2002, p. 17-23.

25. Jerome T. C. Recurrent neural networks and robust time series prediction [Text] / T. C. Jerome, R. M. Douglas, L. E. Atlas // IEEE transactions on neural networks. – Vol. 5, No. 2. –1994. – p. 240–254.

26. Mohsen H. Artificial neural network approach for short term load forecasting for Illam region [Text] / H. Mohsen, S. Yazdan // World Academy of Science, Engineering and Technology 28 2007. – p. 280–284.

27. Amir F. A. A comparison between neural-network forecasting techniques [Text] / F. A. Amir, I. S. Samir – case study: river flow forecasting. // IEEE Transactions on neural networks. –Vol. 10, No. 2. – 1999. – p. 402–409.

28. Back A. D., Wan E. A., Lawrence S., Tsoi A. C. A unifying view of some training algorithms for multilayer perceptrons with FIR filter synapses [Text] / Eds. by J. Vlontzos, J. Hwang, E. Wilson “Neural Networks for Signal Processing 4”. – N.Y.: IEEE Press, 1994. – P. 146 – 154.

29. Бодянский, Е. В. Искусственные нейронные сети: архитектуры, обучение, применения [Текст] / Е. В. Бодянский, О. Г. Руденко // Харьков : ТЕЛЕТЕХ, 2004. – 369 с.

30. Черепанов Ф.М. Искусственный интеллект. Элективный курс: Методическое пособие [Текст] / Черепанов Ф.М., Ясницкий Л.Н.,– М.: БИНОМ. Лаборатория знаний, 2011. – 216с.

31. Попов Э.В. Искусственный интеллект: В 3 кн. Кн. 1. Системы общения и экспертные системы: Справочник [Текст] / Под ред. Э. В. Попова.—М.: Радио и связь, 1990.—464 с.

32. Пилиньский М. Нейронные сети, генетические алгоритмы и нечеткие системы [Текст] / Пилиньский М. Рутковская Д.,, Рутковский Л. - М.: Горячая линия - Телеком, 2006, 452 с.

33. Зайченко Ю.П. Нечеткие модели и методы в интеллектуальных системах: Учеб. пособие [Текст] / Зайченко Ю. П. – К.: Слово, 2008. – 341с.

34. Галямина И.Г. Управление процессами: Учебник для вузов. Стандарт третьего поколения [Текст] / И.Г. Галямина. –СПб.: Питер, 2013. –304 с.

35. Репин В.В., Елиферов В.Г. Процессный подход к управлению. Моделирование бизнес-процессов [Текст] / В.В. Репин, В.Г. Елиферов. – М.: Мани, Иванов и Фербер, 2013. –544 с.

36. Леоненков А.В. Самоучитель UML 2 [Текст] / А.В. Леоненков. –СПб.: БХВ-Петербург, 2007. –576с.

37. Колесов Ю.Б. Моделирование систем. Объектно-ориентированный подход. Учебное пособие [Текст] / Ю.Б. Колесов, Ю.Б Сениченков. -СПб.: БХВ-Петербург, 2012. - 192 с.

38. Шмуллер Джозеф. Освой самостоятельно UML за 24 часа, 3-е издание [Текст] / Д. Шмуллер. – М.: Издательский дом "Вильямс", 2005. – 416с.

39. Дорот В.Л., Новиков Ф.А. Толковый словарь современной компьютерной лексики. - 3-е изд., перераб. и доп [Текст] / В.Л. Дорот, Ф.А. Новиков – СПб.: БХВ-Петербург, 2004. - 608 с.

40. Основні елементи і поняття IDEF0 [Електронний ресурс]. - Режим доступу : URL : http://um.co.ua/3/3-13/3-132841.html

41. Andrew Filev Professional UML Using Visual Studio .Net; Publilkat - Москва, 2012. - 360 с.

42. Нотация IDEF0 [Електронний ресурс] – Режим доступу до ресурсу: http://www.businessstudio.ru/wiki/docs/v4/doku.php/ru/csdesign/bpmodeling/idef0.

43. Диаграмма вариантов использования (use case diagram) [Електронний ресурс] – Режим доступу до ресурсу: http://www.info-system.ru/designing/methodology/uml/theory/use_case_diagram_theory.html.

44. Диаграмма состояний (statechart diagram) [Електронний ресурс] – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1007/229/lecture/5954?page=4.

45. Диаграммы кооперации и их нотация [Електронний ресурс] – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1007/229/lecture/5960?page=3.

46. Диаграмма последовательностей (sequence diagram) [Електронний ресурс] – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1007/229/lecture/5954?page=3.

47. Диаграмма активностей [Електронний ресурс] – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1007/229/lecture/5958.

48. Fainzilberg L.S.Comparative evaluation of convergence's speed of learning algorithms for linear classifiers by statistical experiments method introduction [Text] / L.S. Fainzilberg, N.A. Matushevych // Кибернетика и вычислительная техника. – 2018. – № 2. – 17 p.

Додаток А

Лістинг коду

```
namespace Perceptron
{
        public partial class frmMain : Form
        {
                List<Sample> samples = new List<Sample>();
                List<Sample> points = new List<Sample>();
                List<Sample> audit = new List<Sample>();

                List<Point> linesPerceptron = new List<Point>();
                List<Point> linesKozinets = new List<Point>();
        StreamWriter qqqqqqq;
        SaveFileDialog saveFileDia;

        Graphics objGraphics, objGraphics2;
                Graphics grph, grph2, formGraphics;
                double w1, w2, w0 = 0.0;
                double alpha = 0.5;
                double x0 = -1.0;
                int iterations = 0;

                double x1, x2, y1, y2;
                int cl1, cl2;
                public static int drawingSpeed = 0, learningRate = 250;
                public static bool checkSpeed = false;
                public static bool drawGenLine = false;
                public static int numExpr = 100;
        public static int[] arrIterPerceptron, arrIterKozinets;
        static public bool chartFlag = true;
        static public bool firstKozinetsPoints = false;
        static public bool saveKbKozinets = false;
        static public bool saveKbPerceptron = false;
        int iterPerceptron, iterKozinets, globalIterPerceptron, globalIterKozinets, globalDraw = 0;
        double duration;
        double
                            xStart = 0,
                            yStart = 0,
                            xEnd = 0,
                            yEnd = 0;

                int maxIterations;
                public frmMain()
                {
                        InitializeComponent();
                        //AutoScaleMode = AutoScaleMode.Dpi;
                        //trackLearningRate.Value = 250;
                }
                private void frmMain_Load(object sender, EventArgs e)
                {

                        objGraphics = pnlCanvas.CreateGraphics();
                        grph = panel3.CreateGraphics();
                        grph2 = panel4.CreateGraphics();
                        objGraphics2 = pnlCanvas2.CreateGraphics();

                iterations = 0;
                label1.Visible = true;
```

```
                int i;
                bool error = true;
                maxIterations = 100000;
                Random rnd = new Random();
                w0 = 0;
                w1 = 0;
                w2 = 1;
                alpha = (double)frmMain.learningRate / 1000;
                while (error && iterations < maxIterations)
                {
                    error = false;

                    for (i = 0; i <= samples.Count - 1; i++)
                    {
                        double x1 = samples[i].X1;
                        double x2 = samples[i].X2;
                        int y;

                        if (((w1 * x1) + (w2 * x2) + (w0 * x0)) < 0)
                        {
                            y = -1;
                        }
                        else
                        {
                            y = 1;
                        }

                        if (y != samples[i].Class)
                        {
                            error = true;

                            w0 = w0 + alpha * (samples[i].Class - y) * x0; //ak by x0 bolo +1, tak nedelime
2
                            w1 = w1 + alpha * (samples[i].Class - y) * x1;
                            w2 = w2 + alpha * (samples[i].Class - y) * x2;
                            iterations++;
                            if (checkSpeed)
                                DrawSeparationLine();
                            if (saveKbPerceptron)
                            {
                                qqqqqqq.WriteLine(0 + "\t" + -w1 / w2 + "\t" + -(x0 * w0) / w2);
                            }
                        }
                    }
            }
                         iterPerceptron = iterations;
                         objGraphics.Clear(Color.White);
                         objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2),
new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                         objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));
                         DrawSamples();
                         if (drawGenLine)
                                 objGraphics.DrawLine(new Pen(Color.Gray), new Point((int)xStart,
(int)yStart), new Point((int)xEnd, (int)yEnd));
                         DrawSeparationLine();
                }
                private void kozinets()
                {
            DateTime Start; // Время запуска
            DateTime Stoped; //Время окончания
```

```
            TimeSpan Elapsed = new TimeSpan(); // Разница
            Start = DateTime.Now; // Старт (Записываем время)
            int kozinetsIterations = 0;
            iterKozinets = 0;
            audit.Clear();
                        List<int> class1 = new List<int>();
                        List<int> class2 = new List<int>();
                        int counter = 0;
                        foreach (Sample a in samples)
                        {
                                if (a.Class == 1)
                                        class1.Add(counter++);
                                else
                                        class2.Add(counter++);
                        }

                        Random random = new Random();

            counter = 0;
            while ((class1.Count == 0 || class2.Count == 0) || ((class1.Count < 5 && class2.Count < 5) &&
Math.Abs(class1.Count - class2.Count) > 10))
            {
                class1.Clear();
                class2.Clear();
                counter = 0;
                autoGereration();
                perceptron();
                                foreach (Sample a in samples)
                                {
                                        if (a.Class == 1)
                                                class1.Add(counter++);
                                        else
                                                class2.Add(counter++);
                                }
                        }

            int indForClass1 = random.Next(class1.Count);
            int indForClass2 = random.Next(class2.Count);


            x1 = samples[class1[indForClass1]].X1;
                        x2 = samples[class2[indForClass2]].X1;
                        y1 = samples[class1[indForClass1]].X2;
                        y2 = samples[class2[indForClass2]].X2;


                double posX1 = (pnlCanvas.Width / 2) + (x1) * 10;
                double posY1 = (pnlCanvas.Height / 2) - (y1) * 10;
                double posX2 = (pnlCanvas.Width / 2) + (x2) * 10;
                double posY2 = (pnlCanvas.Height / 2) - (y2) * 10;


                Rectangle rect = new Rectangle((int)posX1 - 2, (int)posY1 - 2, 6, 6);
                Rectangle rect2 = new Rectangle((int)posX2 - 2, (int)posY2 - 2, 6, 6);



                        Kozinets k = new Kozinets(samples);
                        Sample s = new Sample(0,0,-1);
                        double xSredPerpend1 = -10, xSredPerpend2 = 10;
```

```
                double ySredPerpend1=0;
                double ySredPerpend2=0;

                Pen pen = new Pen(Color.Gray);

            while (s != null)
                {
                        kozinetsIterations++;
                        Package p = k.execute(x1,x2,y1,y2);
                        if (checkSpeed)
                                drawSingleLine(p.x1, p.x2, p.y1, p.y2, Color.Purple); //first
random line
                        linesKozinets.Add(new Point((int)p.x1, (int)p.y1));
                        linesKozinets.Add(new Point((int)p.x2, (int)p.y2));

                        ySredPerpend1 = p.kPerp * xSredPerpend1 + p.bPerp;
                        ySredPerpend2 = p.kPerp * xSredPerpend2 + p.bPerp;

                        if (checkSpeed)
                                drawSingleLine(xSredPerpend1, xSredPerpend2, ySredPerpend1,
ySredPerpend2, Color.Black); //perpen

                        linesKozinets.Add(new Point((int)xSredPerpend1, (int)ySredPerpend1));
                        linesKozinets.Add(new Point((int)xSredPerpend2, (int)ySredPerpend2));
                        s = checkKozinets(p.kPerp, p.bPerp);
            if (saveKbKozinets)
            {
                qqqqqqq.WriteLine(1 + "\t" + p.kPerp + "\t" + p.bPerp);
            }

            if (s!=null && s.Class == 999)
            {
                class1.Clear();
                class2.Clear();
                Thread.Sleep(5);
               // Console.WriteLine("new kozinets");
                kozinets();
                return;
            }
                        iterKozinets = kozinetsIterations;
                        objGraphics2.Clear(Color.White);
                        objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height /
2), new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                        objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));
                        DrawSamples();
                        drawSingleLine(xSredPerpend1, xSredPerpend2, ySredPerpend1, ySredPerpend2,
Color.Red);
                        if (drawGenLine)
                                objGraphics2.DrawLine(new Pen(Color.Gray), new Point((int)xStart,
(int)yStart), new Point((int)xEnd, (int)yEnd));

            if (firstKozinetsPoints)
            {

                objGraphics2.DrawEllipse(new Pen(Color.Black), rect);

                objGraphics2.DrawEllipse(new Pen(Color.Black), rect2);

            }
```

```
            label2.Text = "Число итераций: " + kozinetsIterations;
                        label2.Visible = true;

            Stoped = DateTime.Now; // Стоп (Записываем время)
            Elapsed = Stoped.Subtract(Start); // Вычитаем из Stoped (когда код выполнился) время Start
(когда код запустили на выполнение)
            duration = Convert.ToDouble(Elapsed.TotalMilliseconds);
        }
                private void calculateNewPoints(Sample s)
                {
                        double k = 0, b = 0, nx, ny;

                        if (s.Class == 1)
                        {
                                k = (s.X2 - y1) / (s.X1 - x1);
                                b = -(x1 * s.X2 - s.X1 * y1) / (s.X1 - x1); //уравнение прямой между
предыдущей главной (x1,y1) и ошибочно определенной

                                nx = -(y2 - b + x2/k) / (-k - 1/k);
                                ny = k * nx + b;
                             // if (nx < Math.Min(x1, s.X1) || ny < Math.Min(y1, s.X2) || nx > Math.Max(x1,
s.X1) || ny > Math.Max(y1, s.X2))
                                if (!(nx > Math.Min(x1, s.X1) && nx < Math.Max(x1, s.X1) && ny >
Math.Min(y1, s.X2) && ny < Math.Max(y1, s.X2)))
                                {
                                        x1 = s.X1;
                                        y1 = s.X2;
                                }
                                else
                                {
                                        x1 = nx;
                                        y1 = ny;
                                        ////Console.WriteLine("SYUDA NE ZAHODIT");
                                }
                        }
                        else
                        {
                                k = (y2 - s.X2) / (x2 - s.X1);
                                b = -(s.X1 * y2 - x2 * s.X2) / (x2 - s.X1); //уравнение прямой между
предыдущей главной (x2,y2) и ошибочно определенной

                                nx = -(y1 - b + x1 / k) / (-k - 1 / k);
                                ny = k * nx + b;
                                //if (nx < Math.Min(x1, x2) || ny < Math.Min(y1, y2) || nx > Math.Max(x1,
x2) || ny > Math.Max(y1, y2))
                                if (!(nx > Math.Min(x2, s.X1) && nx < Math.Max(x2, s.X1) && ny >
Math.Min(y2, s.X2) && ny < Math.Max(y2, s.X2)))
                                {
                                        x2 = s.X1;
                                        y2 = s.X2;
                                }
                                else
                                {
                                        x2 = nx;
                                        y2 = ny;
                                }
                        }
                        //Console.WriteLine("nx: " + nx + " ny: " + ny);

                }
```

```
        private Sample checkKozinets(double k, double b)
        {

                Random rnd = new Random();
                int cl1Less=0, cl1More=0, cl2Less=0, cl2More=0;
                List<Sample> class1 = new List<Sample>();
                List<Sample> class2 = new List<Sample>();
                foreach (Sample a in samples)
                {
                        if (a.Class == 1)
                                class1.Add(a);
                        else
                                class2.Add(a);
                }


                Sample result = null;

                while (class1.Count > 0)
                {
                        int index = rnd.Next(0, class1.Count - 1);
            label5.Text = "index " + index + "        count " ;
            Sample sample = class1[index];
        if (result != null && result.Class != 999)
        {
            audit.Add(result);
        }

                return result;
        }

        private void drawSingleLine(double x1, double x2, double y1, double y2, Color color)
        {
                double posX1 = (pnlCanvas.Width / 2) + x1 * 10;
                double posY1 = (pnlCanvas.Height / 2) - y1 * 10;

                double posX2 = (pnlCanvas.Width / 2) + x2 * 10;
                double posY2 = (pnlCanvas.Height / 2) - y2 * 10;


                Pen pen = new Pen(color);

                objGraphics2.DrawLine(pen, new Point((int)posX1, (int)posY1), new
Point((int)posX2, (int)posY2));
                //grph2.DrawLine(pen, new Point((int)posX1, (int)posY1), new Point((int)posX2,
(int)posY2));
        }

        private void drawIterChart()
        {
                /*chart1.Series[0].Points.Clear();
                string[] seriesArray = { "Perceptron", "Kozinets" };
                int[] pointsArray = { iterPerceptron, iterKozinets};

                this.chart1.Series["Series1"].Points.DataBindXY(seriesArray, pointsArray);*/

                float x1, y1, x2, y2, hight1, hight2;
                float graphArea = 150;
                float max = Math.Max(iterKozinets, iterPerceptron);
                SolidBrush brush;
```

```
                    grph.Clear(Color.White);

                    x1 = (float)(pnlCanvas.Width / 8);
                    hight1 = iterPerceptron * graphArea / max;
                    y1 = panel3.Height - hight1 - (float)(pnlCanvas.Width / 8);

                    x2 = (float)(5 * pnlCanvas.Width / 8);
                    hight2 = iterKozinets * graphArea / max;
                    y2 = panel3.Height - hight2 - (float)(pnlCanvas.Width / 8);
                    brush = new SolidBrush(Color.Blue);
                    grph.DrawRectangle(new Pen(Color.Red), x1, y1, 50, hight1);
            brush = new SolidBrush(Color.Red);
            grph.FillRectangle(brush, x1, y1, 50, hight1);
            grph.DrawString("Персептрон", this.Font, Brushes.Black, x1 - 3, panel3.Height - 20);
                    grph.DrawString(iterPerceptron.ToString(), this.Font, Brushes.Black, x1 + 20, y1 -
15);
                    grph.DrawRectangle(new Pen(Color.Red), x2, y2, 50, hight2);
            brush = new SolidBrush(Color.Red);
            grph.FillRectangle(brush, x2, y2, 50, hight2);
            grph.DrawString("Козинец", this.Font, Brushes.Black, x2 + 3, panel3.Height - 20);
                    grph.DrawString(iterKozinets.ToString(), this.Font, Brushes.Black, x2 + 20, y2 -
15);
                }
                private void drawGlobalChart()
                {
                    /*chart1.Series[0].Points.Clear();
                    string[] seriesArray = { "Perceptron", "Kozinets" };
                    int[] pointsArray = { iterPerceptron, iterKozinets};

                    this.chart1.Series["Series1"].Points.DataBindXY(seriesArray, pointsArray);*/

                    float x1, y1, x2, y2, x3, y3, hight1, hight2, hight3;
                    float graphArea = 150;
            float max;// = Math.Max(globalIterKozinets, globalIterPerceptron);
                    SolidBrush brush;
            double percentKozinets;
            double percentPerceptron;
            double percentDraw;

            percentKozinets = globalIterKozinets * 100 / (double)(globalIterKozinets + globalIterPerceptron
+ globalDraw);
            percentPerceptron = globalIterPerceptron * 100 / (double)(globalIterKozinets +
globalIterPerceptron + globalDraw);
            percentDraw = globalDraw * 100 / (double)(globalIterKozinets + globalIterPerceptron +
globalDraw);

            max = (int)Math.Max(percentKozinets, percentPerceptron);
            if (max == 0)
                max = (int)percentDraw;

            grph2.Clear(Color.White);

                    x1 = (float)(0.5 * pnlCanvas.Width / 5);
            if (max != 0)
                hight1 = (int)percentPerceptron * graphArea / max;
            else
                hight1 = 0;
            y1 = panel4.Height - hight1 - (float)(pnlCanvas.Width / 8);

                    x2 = (float)(3.5 * pnlCanvas.Width / 5);
```

```
            if (max != 0)
                hight2 = (int)percentKozinets * graphArea / max;
            else
                hight2 = 0;
                        y2 = panel4.Height - hight2 - (float)(pnlCanvas.Width / 8);

            x3 = (float)(2 * pnlCanvas.Width / 5);

            if (max != 0)
                hight3 = (int)percentDraw * graphArea / max;
            else
                hight3 = 0;
            y3 = panel4.Height - hight3 - (float)(pnlCanvas.Width / 8);
            brush = new SolidBrush(Color.Blue);
                        grph2.DrawRectangle(new Pen(Color.Red), x1, y1, 40, hight1);
            brush = new SolidBrush(Color.Red);
            grph2.FillRectangle(brush, x1, y1, 40, hight1);
            grph2.DrawString("Персептрон", this.Font, Brushes.Black, x1 - 10, 7 * panel4.Height / 8);
                        grph2.DrawString(percentPerceptron.ToString("0.0")+ "%", this.Font, Brushes.Black,
x1 + 10, y1 - 15);

                        grph2.DrawRectangle(new Pen(Color.Red), x2, y2, 40, hight2);
            brush = new SolidBrush(Color.Red);
            grph2.FillRectangle(brush, x2, y2, 40, hight2);
            grph2.DrawString("Козинец", this.Font, Brushes.Black, x2 - 3, 7 * panel4.Height / 8);
                        grph2.DrawString(percentKozinets.ToString("0.0") + "%", this.Font, Brushes.Black,
x2 + 10, y2 - 15);

            grph2.DrawRectangle(new Pen(Color.Red), x3, y3, 40, hight3);
            brush = new SolidBrush(Color.Red);
            grph2.FillRectangle(brush, x3, y3, 40, hight3);
            grph2.DrawString("Равный", this.Font, Brushes.Black, x3, 7 * panel4.Height / 8);
            grph2.DrawString("результат", this.Font, Brushes.Black, x3 - 7, 7 * panel4.Height / 8 + 10);
            grph2.DrawString(percentDraw.ToString("0.0") + "%", this.Font, Brushes.Black, x3 + 10, y3 -
15);
        }
                private void all()
                {

                        perceptron();
                        kozinets();
                        drawIterChart();

                }
                private void btnLearn_Click(object sender, EventArgs e)
                {
            if (saveKbKozinets || saveKbPerceptron)
            {
                qqqqqqq = new StreamWriter(saveFileDia.FileName);
                all();
                qqqqqqq.Close();
            }
            else
                all();
                        //txtIterations.Text = iterations.ToString();
                        //txtIterations.Visible = true;
                }

                private void picCanvas_MouseMove(object sender, MouseEventArgs e)
                {
```

```
                double posX = (double)(e.X - pnlCanvas.Width / 2) / 10;
                double posY = (double)(pnlCanvas.Height / 2 - e.Y) / 10;

                this.Text = posX + " " + posY;
            }
            private void btnClear_Click(object sender, EventArgs e)
            {
                samples.Clear();
                objGraphics.Clear(Color.White);
                objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2),
new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));

                objGraphics2.Clear(Color.White);
                objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height /
2), new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));
            }

            private void DrawSamples()
            {
                foreach (Sample sample in samples)
                {

                    double posX = (pnlCanvas.Width / 2) + (sample.X1) * 10;
                    double posY = (pnlCanvas.Height / 2) - (sample.X2) * 10;

                    Pen pen;
                    SolidBrush brush;
                    Rectangle rect = new Rectangle((int)posX - 2, (int)posY - 2, 4, 4);
                    if (sample.Class == 1)
                    {
                        pen = new Pen(Color.Blue);

                        objGraphics.DrawEllipse(pen, rect);
                        objGraphics2.DrawEllipse(pen, rect);

                        brush = new SolidBrush(Color.Blue);
                        objGraphics.FillEllipse(brush, rect);
                        objGraphics2.FillEllipse(brush, rect);
                    }
                    else
                    {
                        pen = new Pen(Color.Red);
               objGraphics.DrawEllipse(pen, rect);
               objGraphics2.DrawEllipse(pen, rect);
               //objGraphics.DrawRectangle(pen, rect);
               //objGraphics2.DrawRectangle(pen, rect);

               /*brush = new SolidBrush(Color.Red);
                        objGraphics.FillRectangle(brush, rect);
                        objGraphics2.FillRectangle(brush, rect);*/
            }


                }
```

```
        }

    private void ChartDraw_Click(object sender, EventArgs e)
    {
        new Chart().ShowDialog();
    }

    private void сохранитьToolStripMenuItem_Click(object sender, EventArgs e)
    {
        string path = null;
        // Configure save file dialog box
        SaveFileDialog saveFileDialog1 = new SaveFileDialog();
        saveFileDialog1.Filter = "Text documents (.txt)|*.txt";
        saveFileDialog1.Title = "Сохранить в файл";
        saveFileDialog1.ShowDialog();
        if (saveFileDialog1.FileName != "")
        {
            // Saves the Image via a FileStream created by the OpenFile method.
            //System.IO.FileStream sw = (System.IO.FileStream)saveFileDialog1.OpenFile();

            StreamWriter sw = new StreamWriter(saveFileDialog1.FileName);

            //string[] st = new string[] { "fff", "aaa", "kkk" };

            foreach (Sample sample in samples)
            {
                // MessageBox.Show(node);
                sw.WriteLine(sample.X1 + " " + sample.X2 + " " + sample.Class);
                //sw.Close();
            }
            MessageBox.Show("Точки сохранены.");
            sw.Close();
        }
    }

    private void открытьToolStripMenuItem_Click(object sender, EventArgs e)
    {
        OpenFileDialog saveFileDialog1 = new OpenFileDialog();
        saveFileDialog1.Filter = "Text documents (.txt)|*.txt";
        saveFileDialog1.Title = "Открыть файл";
        saveFileDialog1.ShowDialog();
        if (saveFileDialog1.FileName != "")
        {
            samples.Clear();
            objGraphics.Clear(Color.White);
            objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2), new
Point(pnlCanvas.Width, pnlCanvas.Height / 2));
            objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0), new
Point(pnlCanvas.Width / 2, pnlCanvas.Height));

            objGraphics2.Clear(Color.White);
            objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2), new
Point(pnlCanvas.Width, pnlCanvas.Height / 2));
            objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0), new
Point(pnlCanvas.Width / 2, pnlCanvas.Height));
            grph.Clear(Color.White);
            grph2.Clear(Color.White);
            //List<Sample> samples = new List<Sample>();
            try
            {
```

```
                    /*
                    if (File.Exists(path))
                    {
                        File.Delete(path);
                    }*/

                    /* using (StreamWriter sw = new StreamWriter(path))
                     {
                         sw.WriteLine("This");
                         sw.WriteLine("is some text");
                         sw.WriteLine("to test");
                         sw.WriteLine("Reading");
                     }
                     */
                    // string s = "";
                    using (StreamReader sr = new StreamReader(saveFileDialog1.FileName))
                    {
                        while (sr.Peek() >= 0)
                        {
                            string[] split = sr.ReadLine().Split(new Char[] { ' ' });
                            samples.Add(new Sample(Convert.ToDouble(split[0]), Convert.ToDouble(split[1]),
Convert.ToInt32(split[2])));
                        }
                    }
                    MessageBox.Show("Точки считаны");
                    DrawSamples();
                }
                catch (Exception ee)
                {
                    ////Console.WriteLine("The process failed: {0}", ee.ToString());
                }
            }
        }

        private void очиститьToolStripMenuItem_Click(object sender, EventArgs e)
        {
            samples.Clear();
            objGraphics.Clear(Color.White);
            objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2), new
Point(pnlCanvas.Width, pnlCanvas.Height / 2));
            objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0), new
Point(pnlCanvas.Width / 2, pnlCanvas.Height));

            objGraphics2.Clear(Color.White);
            objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2), new
Point(pnlCanvas.Width, pnlCanvas.Height / 2));
            objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0), new
Point(pnlCanvas.Width / 2, pnlCanvas.Height));

            grph.Clear(Color.White);
            grph2.Clear(Color.White);
        }

        private void ручнойToolStripMenuItem_Click(object sender, EventArgs e)
        {
            samples.Clear();
            objGraphics.Clear(Color.White);
            objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2), new
Point(pnlCanvas.Width, pnlCanvas.Height / 2));
            objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0), new
Point(pnlCanvas.Width / 2, pnlCanvas.Height));
```

```
            objGraphics2.Clear(Color.White);
            objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2), new
Point(pnlCanvas.Width, pnlCanvas.Height / 2));
            objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0), new
Point(pnlCanvas.Width / 2, pnlCanvas.Height));
            grph.Clear(Color.White);
            grph2.Clear(Color.White);
        }

        private void справкаToolStripMenuItem_Click(object sender, EventArgs e)
        {
            new inform().ShowDialog();
        }

        private void label3_Click_1(object sender, EventArgs e)
                {

                }


                private void настройкиToolStripMenuItem_Click(object sender, EventArgs e)
                {
                        new Settings().ShowDialog();
            button1.Text = "Многократный запуск (" + numExpr.ToString() + ")";
                }


                private void aggiornaContatore()
                {
                        if (this.label1.InvokeRequired)
                        {
                                this.label1.BeginInvoke((MethodInvoker)delegate() { this.label1.Text =
"Число итераций: "+this.iterations.ToString(); ;});
                        }
                        else
                        {
                                this.label1.Text = "Число итераций: "+this.iterations.ToString(); ;
                        }
                }


                private void DrawSeparationLine()
                {
                        Random rnd = new Random();
                        objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2),
new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                        objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));

                        if (!(xStart == 0 && yStart == 0 && xEnd == 0 && yEnd == 0))
                        {
                                if (drawGenLine)
                                {
                                        objGraphics.DrawLine(new Pen(Color.Gray), new Point((int)xStart,
(int)yStart), new Point((int)xEnd, (int)yEnd));
                                        objGraphics2.DrawLine(new Pen(Color.Gray), new Point((int)xStart,
(int)yStart), new Point((int)xEnd, (int)yEnd));
                                }
                        }
```

```
                        Pen pen = new Pen(Color.Red);
                        double x1;

                        x1 = -10;
                        double y = -(x1 * w1 / w2) - ((x0*w0) / w2);

                        double shift = pnlCanvas.Height / 2;


                        // Point p1 = new Point((int)(x1 * 10 + pnlCanvas.Width / 2), (int)(shift - y *
10));
                        Point p1 = new Point((int)(x1 * 10 + pnlCanvas.Width / 2), (int)(shift - y * 10));

                        x1 = 10;
                        double y2 = -(x1 * w1 / w2) - ((x0*w0) / w2);

                        Point p2 = new Point((int)(x1 * 10 + pnlCanvas.Width / 2), (int)(shift - y2 *
10));
                        if (w2 != 0)
                        {
                                objGraphics.DrawLine(pen, p1, p2);
                                points.Add(new Sample(p1.X, p1.Y, 1));
                                points.Add(new Sample(p2.X, p2.Y, -1));
                           // DrawSamples();
                        }
                        Thread.Sleep(drawingSpeed);
                }

                private void menuStrip1_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
                {

                }

                private void speedScroll_Scroll(object sender, ScrollEventArgs e)
                {

                }

                private void trackLearningRate_Scroll(object sender, EventArgs e)
                {
                        /*lblLearningRate.Text = "Коэффициент обучения: " +
(double)trackLearningRate.Value / 1000;
                        String a = "Коэффициент обучения: " + (double)trackLearningRate.Value / 1000;*/
                }

                private void pnlCanvas_MouseDown(object sender, MouseEventArgs e)
                {
                        Sample sample;
                        Pen pen;
                        SolidBrush brush;

                        double posX = (double)(e.X - pnlCanvas.Width / 2) / 10;
                        double posY = (double)(pnlCanvas.Height / 2 - e.Y) / 10;
                        Rectangle rect = new Rectangle((int)e.X - 2, (int)e.Y - 2, 4, 4);

                        if (e.Button == MouseButtons.Left)
                        {
                                pen = new Pen(Color.Blue);
                                sample = new Sample(posX, posY, 1);

                                objGraphics.DrawEllipse(pen, rect);
```

```
                                objGraphics2.DrawEllipse(pen, rect);

                                brush = new SolidBrush(Color.Blue);
                                objGraphics.FillEllipse(brush, rect);
                                objGraphics2.FillEllipse(brush, rect);
                        }
                        else
                        {
                                pen = new Pen(Color.Red);
                                sample = new Sample(posX, posY, -1);

                    objGraphics.DrawEllipse(pen, rect);
                    objGraphics2.DrawEllipse(pen, rect);
                }
                                samples.Add(sample);
                     }

                     private void pnlCanvas_Paint(object sender, PaintEventArgs e)
                     {
                        objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2),
new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                        objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));

                        objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height /
2), new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                        objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));
                     }

                     private void button1_Click(object sender, EventArgs e)
                     {
                ChartDraw.Enabled = true;
                /* btnLearn_Click(null, null);
                 btnLearn_Click(null, null);
                 btnLearn_Click(null, null);*/

                /*
                saveFileDia = new SaveFileDialog();
                saveFileDia.Filter = "Text documents (.txt)|*.txt";
                saveFileDia.Title = "Сохранить в файл";
                saveFileDia.ShowDialog();
                qqqqqqq = new StreamWriter(saveFileDia.FileName);
                */

                globalIterKozinets = 0;
                globalIterPerceptron = 0;
                globalDraw = 0;
                arrIterKozinets = new int[numExpr];
                arrIterPerceptron = new int[numExpr];
               // if (!chartFlag)
                 //   autoGereration();
                for (int i = 0; i < numExpr; i++)
                        //while(true)
                        {
                    if (chartFlag)
                        autoGereration();
                    all();
                    if (iterPerceptron < iterKozinets)
                        globalIterPerceptron++;
                    else if (iterPerceptron > iterKozinets)
```

```
                globalIterKozinets++;
            else
                globalDraw++;
            arrIterPerceptron[i] = iterPerceptron;
            arrIterKozinets[i] = iterKozinets;
                        drawGlobalChart();
                        // Thread.Sleep(1000);
                    }
        //qqqqqqq.Close();
    }

            private void генерацияДанныхToolStripMenuItem_Click(object sender, EventArgs e)
            {
                    /*
                    DataGeneration dataGeneration = new DataGeneration(0,0,4,10);
                    List<Sample> resultSet = dataGeneration.generate();

                    foreach (Sample entry in resultSet)
                    {
                            drawPoints(entry.X1, entry.X2, entry.Class);
                    }
                    DrawSamples(); */
            }


            private void drawHull()
            {
                    ConvexHull c = new ConvexHull(samples);
                    List<int> class1 = c.ConvexHullJarvis(c.class1);
                    List<int> class2 = c.ConvexHullJarvis(c.class2);
                    double posXStart = 0, posYStart = 0, posXFinish = 0, posYFinish = 0;
                    List<Color> colors = new List<Color>();
                    colors.Add(Color.Red);
                    colors.Add(Color.Yellow);
                    colors.Add(Color.Red);
                    colors.Add(Color.Blue);
                    colors.Add(Color.Black);
                    colors.Add(Color.Purple);
                    colors.Add(Color.Violet);

                    for (int i = 1; i < class1.Count; i++)
                    {
                            posXStart = (pnlCanvas.Width / 2) + c.class1[class1[i - 1]].X * 10;
                            posYStart = (pnlCanvas.Height / 2) - c.class1[class1[i - 1]].Y * 10;

                            posXFinish = (pnlCanvas.Width / 2) + c.class1[class1[i]].X * 10;
                            posYFinish = (pnlCanvas.Height / 2) - c.class1[class1[i]].Y * 10;

                            objGraphics2.DrawLine(new Pen(Color.Orange), new Point((int)posXStart,
(int)posYStart), new Point((int)posXFinish, (int)posYFinish));

                    }



                    for (int i = 1; i < class2.Count; i++)
                    {
                            posXStart = (pnlCanvas.Width / 2) + c.class2[class2[i - 1]].X * 10;
                            posYStart = (pnlCanvas.Height / 2) - c.class2[class2[i - 1]].Y * 10;

                            posXFinish = (pnlCanvas.Width / 2) + c.class2[class2[i]].X * 10;
```

```
                                posYFinish = (pnlCanvas.Height / 2) - c.class2[class2[i]].Y * 10;

                                objGraphics2.DrawLine(new Pen(Color.Orange), new Point((int)posXStart,
(int)posYStart), new Point((int)posXFinish, (int)posYFinish));

                        }

                }

                private void drawPoints(double x, double y, double color)
                {

                        Sample sample;
                        Pen pen;

                        double posX = x;
                        double posY = y;

                        if (color==1)
                        {
                                pen = new Pen(Color.Blue);
                                sample = new Sample(posX, posY, 1);
                        }
                        else
                        {
                                pen = new Pen(Color.Red);
                                sample = new Sample(posX, posY, -1);
                        }
                        samples.Add(sample);


                   /*
                        objGraphics.DrawLine(pen, new Point(e.X - 3, e.Y), new Point(e.X + 3, e.Y));
                        objGraphics.DrawLine(pen, new Point(e.X, e.Y - 3), new Point(e.X, e.Y + 3));*/

                }

                public void autoGereration()
                {
                        samples.Clear();
                        objGraphics.Clear(Color.White);
                        objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height / 2),
new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                        objGraphics.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));

                        objGraphics2.Clear(Color.White);
                        objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(0, pnlCanvas.Height /
2), new Point(pnlCanvas.Width, pnlCanvas.Height / 2));
                        objGraphics2.DrawLine(new Pen(Color.Gainsboro), new Point(pnlCanvas.Width / 2, 0),
new Point(pnlCanvas.Width / 2, pnlCanvas.Height));


            DataGeneration dataGeneration = new DataGeneration(0, 0, 15, 30);


                        Dictionary<double, double> line = dataGeneration.generateDataForRandomLine();

                        xStart = -10.0 * 10 + pnlCanvas.Width / 2;
                        yStart = line[10.0] * 10 + pnlCanvas.Width / 2;
                        xEnd = 10.0 * 10 + pnlCanvas.Width / 2;
```

```
                        yEnd = line[-10.0] * 10 + pnlCanvas.Width / 2;

                        if (drawGenLine)
                        {
                                objGraphics.DrawLine(new Pen(Color.Gray), new Point((int)xStart,
(int)yStart), new Point((int)xEnd, (int)yEnd));
                                objGraphics2.DrawLine(new Pen(Color.Gray), new Point((int)xStart,
(int)yStart), new Point((int)xEnd, (int)yEnd));
                        }


                        List<Sample> resultSet = dataGeneration.generateRandomPoints();


                        foreach (Sample entry in resultSet)
                        {
                                if ((-dataGeneration.lineA * entry.X1 + entry.X2 + dataGeneration.lineB) <
-1)
                                        drawPoints(entry.X1, entry.X2, 1);
                                else if ((-dataGeneration.lineA * entry.X1 + entry.X2 +
dataGeneration.lineB) > 1)
                                        drawPoints(entry.X1, entry.X2, -1);
                        }
                        DrawSamples();
                }

                private void автоматическийToolStripMenuItem_Click(object sender, EventArgs e)
                {
            grph.Clear(Color.White);
            grph2.Clear(Color.White);
            autoGereration();
                }

                private void линияГенерацииToolStripMenuItem_Click(object sender, EventArgs e)
                {

                }

                private void label3_Click(object sender, EventArgs e)
                {

                }

                private void trackLearningRate_Scroll_1(object sender, EventArgs e)
                {

                }

        }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Perceptron
{
        public partial class Settings : Form
        {
                public Settings()
                {
                        InitializeComponent();
```

```
                hScrollBar1.Value = (int)DataGeneration.count;
                label3.Text = DataGeneration.count.ToString();

                if (!frmMain.chartFlag)
                        checkTypeSamples.Checked = false;
                else
        checkTypeSamples.Checked = true;

if (!frmMain.firstKozinetsPoints)
        checkBox1.Checked = false;
else
        checkBox1.Checked = true;

if (!frmMain.saveKbPerceptron)
        checkBox2.Checked = false;
else
        checkBox2.Checked = true;

if (!frmMain.saveKbKozinets)
        checkBox3.Checked = false;
else
        checkBox3.Checked = true;

if (!frmMain.drawGenLine)
        checkGenLine.Checked = false;
else
        checkGenLine.Checked = true;

if (frmMain.drawingSpeed == 0)
                {
                        checkSpeed.Checked = false;
                        speedBar.Enabled = false;
                }
else
                {
                        speedBar.Value = frmMain.drawingSpeed;
                        checkSpeed.Checked = true;
                        speedBar.Enabled = true;
                }

                textNameExpr.Text = frmMain.numExpr.ToString();

                trackLearningRate.Value = frmMain.learningRate;
                label5.Text = "" + (double)trackLearningRate.Value / 1000;

        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void label2_Click(object sender, EventArgs e)
        {

        }

        private void label3_Click(object sender, EventArgs e)
        {

        }

        private void hScrollBar1_Scroll(object sender, ScrollEventArgs e)
        {
                label3.Text = hScrollBar1.Value.ToString();
                DataGeneration.count = hScrollBar1.Value;
        }

        private void Settings_Load(object sender, EventArgs e)
        {

        }

        private void label4_Click(object sender, EventArgs e)
        {

        }

        private void label4_Click_1(object sender, EventArgs e)
        {

        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
                if (!checkSpeed.Checked)
                {
                        speedBar.Enabled = false;
                        speedLabel.Text = "";
                        frmMain.checkSpeed = false;
                        frmMain.drawingSpeed = 0;
```

```
                }
                else
                {
                        speedBar.Enabled = true;
                        speedLabel.Text = speedBar.Value.ToString();
                        frmMain.checkSpeed = true;
                        frmMain.drawingSpeed = speedBar.Value;
                }
        }

        private void speedBar_Scroll(object sender, ScrollEventArgs e)
        {
                if (speedBar.Enabled)
                {
                        speedLabel.Text = speedBar.Value.ToString();
                        frmMain.drawingSpeed = speedBar.Value;
                }
                else
                {
                        speedLabel.Text = "0";
                        frmMain.drawingSpeed = 0;
                }
        }

        private void trackLearningRate_Scroll(object sender, EventArgs e)
        {
                frmMain.learningRate = trackLearningRate.Value;
                label5.Text = "" + (double)trackLearningRate.Value / 1000;
        }

        private void checkGenLine_CheckedChanged(object sender, EventArgs e)
        {
                if (!checkGenLine.Checked)
                {
                        frmMain.drawGenLine = false;
                   /* frmMain.DrawSamples();
                        frmMain.DrawSeparationLine();
                        frmMain.objGraphics.DrawLine(new Pen(Color.Gray), new Point((int)xStart,
(int)yStart), new Point((int)xEnd, (int)yEnd));*/
                }
                else
                {
                        frmMain.drawGenLine = true;
                }

        }

        private void textNameExpr_TextChanged(object sender, EventArgs e)
        {

                        frmMain.numExpr = int.Parse(textNameExpr.Text);


        }

        private void textNameExpr_KeyPress(object sender, KeyPressEventArgs e)
        {
                // ввод в texBox только цифр и кнопки Backspace
                char ch = e.KeyChar;
                if (textNameExpr.Text.Length == 0)
                        if (e.KeyChar == '0')
                                e.Handled = true;
                if (!Char.IsDigit(ch) && ch != 8)
                         e.Handled = true;

        }

    private void panel2_Paint(object sender, PaintEventArgs e)
    {

    }

    private void checkTypeSamples_CheckedChanged(object sender, EventArgs e)
    {
        if (!checkTypeSamples.Checked)
        {
            frmMain.chartFlag = false;
        }
        else
            frmMain.chartFlag = true;

    }

    private void checkBox1_CheckedChanged_1(object sender, EventArgs e)
    {
        if (!checkBox1.Checked)
        {
            frmMain.firstKozinetsPoints = false;
        }
        else
            frmMain.firstKozinetsPoints = true;
```

```
        }

        private void checkBox2_CheckedChanged(object sender, EventArgs e)
        {
            if (!checkBox2.Checked)
            {
                frmMain.saveKbPerceptron= false;
            }
            else
                frmMain.saveKbPerceptron = true;
        }

        private void checkBox3_CheckedChanged(object sender, EventArgs e)
        {
            if (!checkBox2.Checked)
            {
                frmMain.saveKbKozinets = false;
            }
            else
                frmMain.saveKbKozinets = true;
        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {

        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace Perceptron
{
    public partial class Chart : Form
    {
        public Chart()
        {
            InitializeComponent();
        }

        private int[] arrSort(int [] nums)
        {
            int temp;
            for (int i = 0; i < nums.Length - 1; i++)
            {
                for (int j = i + 1; j < nums.Length; j++)
                {
                    if (nums[i] > nums[j])
                    {
                        temp = nums[i];
                        nums[i] = nums[j];
                        nums[j] = temp;
                    }
                }
            }
            return (nums);
        }

        static int[] Distinct(int[] a)
        {
            HashSet<int> list = new HashSet<int>(a);
            int[] b = new int[list.Count];
            list.CopyTo(b);

            return b;
        }

        static int DigitCount(int[] a, int b)
        {
            int count = 0;

            foreach (var i in a)
                if (i == b) count++;

            return count;
        }

        private void Chart_Load(object sender, EventArgs e)
        {
            chart4.Visible = false;
            label1.Text = "Гистограмма результатов алгоритма персептрона";
            label2.Text = "Гистограмма результатов алгоритма Козинца";
            if (checkChart.Checked)
```

```
            {
                for (var i = 0; i < frmMain.arrIterPerceptron.Length; i++)
                {
                    chart1.Series[0].Points.AddXY(i + 1, frmMain.arrIterPerceptron[i]);
                    chart2.Series[0].Points.AddXY(i + 1, frmMain.arrIterKozinets[i]);
                }
            }

            int minElement1 = frmMain.arrIterPerceptron[0];
            int maxElement1 = frmMain.arrIterPerceptron[0];
            int sum1 = 0, sum2 = 0;
            double average1 = 0.0;
            double average2 = 0.0;
            double sumDiff1 = 0.0;
            double sumDiff2 = 0.0;
            double dispersion1;
            double dispersion2;
            double coefOfVariation1;
            double coefOfVariation2;

            foreach (int element in frmMain.arrIterPerceptron)
            {
                if (element < minElement1)
                {
                    minElement1 = element;
                }
                else if (maxElement1 < element)
                {
                    maxElement1 = element;
                }
                sum1 += element;
            }

            average1 = sum1 / frmMain.numExpr;

            label3.Text = "Наименьшее количество итераций: " + minElement1.ToString();
            label4.Text = "Наибольшее количество итераций: " + maxElement1.ToString();

            minElement1 = frmMain.arrIterKozinets[0];
            maxElement1 = frmMain.arrIterKozinets[0];

            foreach (int element in frmMain.arrIterKozinets)
            {
                if (element < minElement1)
                {
                    minElement1 = element;
                }
                else if (maxElement1 < element)
                {
                    maxElement1 = element;
                }
                sum2 += element;
            }

            average2 = sum2 / frmMain.numExpr;


            label5.Text = "Наименьшее количество итераций: " + minElement1.ToString();
            label6.Text = "Наибольшее количество итераций: " + maxElement1.ToString();

            for (int i =0; i < frmMain.numExpr; i++)
            {
                sumDiff1 += Math.Pow((frmMain.arrIterPerceptron[i] - average1), 2);
                sumDiff2 += Math.Pow((frmMain.arrIterKozinets[i] - average2), 2);
            }

            dispersion1 = sumDiff1 / frmMain.numExpr;
            dispersion2 = sumDiff2 / frmMain.numExpr;

            coefOfVariation1 = (Math.Sqrt(dispersion1) / average1) * 100;
            coefOfVariation2 = (Math.Sqrt(dispersion2) / average2) * 100;

           // if (coefOfVariation1 < 100)
                label7.Text = "Коэффициент вариации (персептрон): " + coefOfVariation1.ToString("0.0") +
"%";
            /*else
                label7.Text = "Коэффициент вариации (персептрон): 100%";*/
            //if (coefOfVariation2 < 100)
                label8.Text = "Коэффициент вариации (Козинец): " + coefOfVariation2.ToString("0.0") + "%";
           /* else
                label8.Text = "Коэффициент вариации (Козинец): 100%";*/
        }

        private void ChartChecked()
        {
            int[] arrPerceptron = new int[frmMain.numExpr];
            int[] arrKozinets = new int[frmMain.numExpr];

            chart1.Series.Clear();
            chart2.Series.Clear();
```

```
    if (checkChart.Checked)
    {
        chart1.Visible = true;
        chart3.Visible = false;
        chart2.Visible = true;
        chart4.Visible = false;
        for (var i = 0; i < frmMain.arrIterPerceptron.Length; i++)
        {
            chart1.Series[0].Points.AddXY(i + 1, frmMain.arrIterPerceptron[i]);
            chart2.Series[0].Points.AddXY(i + 1, frmMain.arrIterKozinets[i]);
        }
    }
    else
    {
        chart1.Visible = false;
        chart3.Visible = true;
        chart2.Visible = false;
        chart4.Visible = true;
        for (var i = 0; i < frmMain.arrIterPerceptron.Length; i++)
        {
            arrPerceptron[i] = frmMain.arrIterPerceptron[i];
            arrKozinets[i] = frmMain.arrIterKozinets[i];
        }
        arrPerceptron = arrSort(arrPerceptron);
        arrKozinets = arrSort(arrKozinets);

        /* for (var i = 0; i < frmMain.arrIterPerceptron.Length; i++)
         {
            if (i in Distinct(arrPerceptron))
         }*/
        Dictionary<int, int> CountPerc = new Dictionary<int, int>();
        for (int i = 0; i < arrPerceptron.Length;)
        {
            int j;
            for (j = i + 1; j < arrPerceptron.Length; j++)
            {
                if (arrPerceptron[i] != arrPerceptron[j])
                {
                    CountPerc.Add(arrPerceptron[i], j - i);
                    i = j;
                    break;
                }
            }
            if (j == arrPerceptron.Length)
            {
                CountPerc.Add(arrPerceptron[i], j - i);
                break;
            }
        }
        Dictionary<int, int> CountKoz = new Dictionary<int, int>();
        for (int i = 0; i < arrKozinets.Length;)
        {
            int j;
            for (j = i + 1; j < arrKozinets.Length; j++)
            {
                if (arrKozinets[i] != arrKozinets[j])
                {
                    CountKoz.Add(arrKozinets[i], j - i);
                    i = j;
                    break;
                }
            }
            if (j == arrKozinets.Length)
            {
                CountKoz.Add(arrKozinets[i], j - i);
                break;
            }
        }



        int test;
        for (var i = 1; i < frmMain.arrIterPerceptron.Length; i++)
        {
            if (CountPerc.TryGetValue(i, out test))
                chart3.Series[0].Points.AddXY(i, CountPerc[i]);
           /* else
                mySeriesOfPoint.Points.AddXY(i, 0);*/
            if (CountKoz.TryGetValue(i, out test))
                chart4.Series[0].Points.AddXY(i, CountKoz[i]);
           /* else
                chart2.Series[0].Points.AddXY(i, 0);*/
        }
       // chart3.Series.Add(mySeriesOfPoint);
        /*chart1.Series[0].Points.AddXY(num, i);

                 chart2.Series[0].Points.AddXY(i, DigitCount(arrKozinets, i));*/
    }
}

private void checkChart_CheckedChanged(object sender, EventArgs e)
```

```
            {
                ChartChecked();
            }
        }
    }
```